Proyecto fin de carrera

Tema: Técnicas de Procesado Digital de la Señal.

Título: Diseño y análisis de unidades didácticas para Procesado Digital de la Señal.

Autor: Mario Jiménez Recuero.

Titulación: Sistemas Electrónicos.

Tutor: D. César Benavente Peces.

Departamento: Departamento de Teoría de la Señal y Comunicaciones.

Secretario: D. Luis Arriero Encinas.

Secretario vocal: D. César Benavente Peces.

Presidente: D. Constantino Gil González.

Fecha de lectura: 29 de septiembre de 2014.

Quiero dedicar este proyecto, en primer lugar a mis padres, ya que todo lo que soy es gracias a ellos. Sé que desde el cielo estarán viviendo con alegría este momento conmigo.

En segundo lugar a Toñi, mi mujer, sin ella nunca habría llegado hasta aquí. Gracias por estar siempre a mi lado.

En tercer lugar a Mateo, mi hijo, por darme ese último empujón que necesitaba para poder acabar este proyecto.

Por último quiero agradecer a César, mi tutor, su tiempo y su dedicación.

"No se puede pactar con las dificultades, o las vencemos o nos vencen"

Índice

Resumen

Este Proyecto Fin de Carrera pretende desarrollar una serie de unidades didácticas orientadas a mejorar el aprendizaje de la teoría de procesado digital de señales a través de la aplicación práctica. Con tal fin, se han diseñado una serie de prácticas que permitan al alumno alcanzar un apropiado nivel de conocimiento de la asignatura, la adquisición de competencias y alcanzar los resultados de aprendizaje previstos.

Para desarrollar el proyecto primero se ha realizado una selección apropiada de los contenidos de la teoría de procesado digital de señales en relación con los resultados de aprendizaje esperados, seguidamente se han diseñado y validado unas prácticas basadas en un entorno de trabajo basado en MATLAB y DSP, y por último se ha redactado un manual de laboratorio que combina una parte teórica con su práctica correspondiente.

El objetivo perseguido con la realización de estas prácticas es alcanzar un equilibrio teórico/práctico que permita sacar el máximo rendimiento de la asignatura desde el laboratorio, trabajando principalmente con el IDE Code Composer Studio junto con un kit de desarrollo basado en un DSP.

Abstract

This dissertation intends to develop some lessons oriented to improve about the digital signal processing theory. In order to get this objective some practices have been developed to allow to the students to achieve an appropriate level of knowledge of the subject, acquire skills and achieve the intended learning outcomes.

To develop the project firstly it has been made an appropriate selection of the contents of the digital signal processing theory related with the expected results. After that, five practices based in a work environment based on Matlab and DSP have been designed and validated, and finally a laboratory manual has been drafted that combines the theoretical part with its corresponding practice.

The objective with the implementation of these practices is to achieve a theoretical / practical balance to get the highest performance to the subject from the laboratory working mainly with the Code Composer Studio IDE together a development kit based on DSP.

1 Introducción

1.1 Objetivo

Este proyecto surge con la finalidad de complementar y mejorar el aprendizaje de los alumnos de la asignatura de Procesado Digital de Señales.

El objetivo es proporcionar un entorno de trabajo basado en DSP con el que los alumnos puedan reforzar y poner en práctica los conocimientos adquiridos en la parte teórica de la asignatura.

1.2 Organización y alcance del proyecto

El trabajo realizado se ha dividido en tres fases. La primera fase ha consistido en seleccionar los contenidos teóricos de la asignatura. Estos contenidos han sido organizados en cinco unidades didácticas de la siguiente manera:

- Introducción al entorno de desarrollo: el objetivo es conocer y familiarizarse con los elementos tanto software como hardware que se van a utilizar en el laboratorio.
- Muestreo: es la primera práctica basada en contenidos propios de la asignatura. El alumno debe ser capaz de anticipar el comportamiento del sistema en base a las diferentes configuraciones del hardware utilizado para el procesado.
- Filtrado FIR: con esta práctica el alumno tendrá la oportunidad de implementar en el DSP un filtro FIR caracterizado por el mismo con ayuda de la herramienta Matlab.
- Filtrado IIR: con un desarrollo gemelo a la práctica anterior. Además con esta práctica el alumno podrá comparar las limitaciones de uno y otro a la hora de implementarlos en un sistema real.
- Transformada discreta de Fourier: permite obtener un representación en el dominio de la frecuencia, siendo la función original una función en el dominio del tiempo. Este es el último de los contenidos elegidos para poder completar el aprendizaje de la asignatura.

Una vez hecha la selección de contenidos,la segunda fase ha sido definir un

entorno de trabajo basado en DSP y Matlab que permita alcanzar los objetivos marcados de una manera clara y sencilla, por eso y por su bajo coste se ha elegido el kit de desarrollo de Texas Instruments TMX320C5515 eZDSP v2 USB STICK que incluye el IDE CodeComposer Studio y que nos ofrece todo lo necesario y mucho más para poder alcanzar los objetivos marcados anteriormente.

De esta manera se le ofrece la posibilidad a los alumnos de poder trabajar con un entorno de procesado digital de señales en tiempo real, de forma que puedan ser capaces de valorar y experimentar todos los factores necesarios a la hora de poner en funcionamiento un sistema de esta naturaleza.

Como se ha comentado anteriormente, otro de los factores que havalorado es el económico. Por eso, el sistema propuesto para el desarrollo de las prácticas no va a suponer un gasto importante ni a los alumnos ni a la Universidad.

Para el correcto desarrollo del laboratorio serán necesarios algunos elementos tanto hardware como software, algunos de ellos han sido nombrados en líneas anteriores, pero a continuación se verá cada uno de manera breve, ya que más adelante hay una descripción detallada de cada uno de ellos:

- El elemento principal sobre el cual se basa todo el trabajo realizado en este proyecto, es un kit de desarrollo del fabricante Texas Instruments. Se trata del TMX320C5515 eZDSP v2 USB STICK. La tarjeta incluida en el kit tiene la gran ventaja de que puede ser alimentada directamente a través de un puerto USB del PC que permite que sea muy fácil poder trabajar con ella, ya que será este mismo puerto el que utilizaremos para la carga de los programas en la tarjeta. Además del puerto USB, la tarjeta contiene dos puertos de audio estéreoque se utilizan para introducirle las señales al DSP y sacar el resultado procesado.
- Para poder conectar la tarjeta al PC, además del cable USB que incluye el propio kit, será necesario disponer de dos cables estéreo con conector mini jack en cada uno de sus extremos. Estos cables serán empleados para interconectar la salida de audio del PC con la tarjeta y la salida de la tarjeta con la entrada de audio del PC, haciendo uso de los puertos de la tarjeta de sonido del PC.

- Además, el kit incluye el IDE CodeComposer Studio, softwarepropietario de Texas Instruments. Esta herramienta es lautilizada para la programación y configuración del DSP y sus periféricos.
- Otra de las herramientas software necesarias es SoundcardScope, se trata de un programa de libre distribución para usos educativos. Este software implementa con ayuda de la tarjeta de sonido del PC un osciloscopio, un generador de señales, un analizador de frecuencias y alguna otra función que será detallada en su apartado correspondiente.
- Con todo esto ya solo falta una herramienta que nos ayude a la hora de calcular y caracterizar nuestras funciones, en nuestro caso tanto los filtros FIR como los IIR. Esa herramienta no podía ser otra que Matlab con su módulo específico de procesado digital de señales, con él, serán calculados y caracterizados los filtros que más tarde se exportarán a CodeComposer para su implementación en el DSP.

Por último, la tercera fase ha consistido en la programación del DSP. Esta fase tenía dos objetivos, el primero era comprobar que todo aquello que inicialmente se había pensado hacer con en el DSP era posible hacerlo. Y el segundo, era poder entregar a los alumnos una serie de proyectos de CCS lo más completos posible, para las distintas prácticas, de manera que el alumno no debería malgastar tiempo resolviendo cuestiones relacionadas con la programación, ya que no es uno de los objetivos perseguidos en la asignatura.

Con estas herramientas y el manual de las prácticas, el alumno, una vez haya superado de forma satisfactoria el laboratorio, debería haber alcanzado unos conocimientos acordes a los objetivos de la asignatura. Así mismo será capaz de comprender el funcionamiento en todas sus fases, tanto diseño como implementación, de un sistema de procesado de señales en tiempo real.

2 Descripción del hardware y software empleado:

En esta sección se revisará de una manera más detallada todos los elementos que hemos puesto en funcionamiento para poder alcanzar los objetivos marcados.

En primer lugar se ha estudiado el hardware empleado, prestando especial atención en aquellos elementos que se consideran esenciales y a continuación , y por último revisaremos todos los contenidos teóricos del procesado digital de señales y como los hemos implementado en nuestro sistema de DSP.

2.1 Análisis del hardware empleado.

El hardware elegido para la realización del proyecto ha sido el kit de desarrollo de Texas Instruments, TMX320C5515 eZdsp USB STICK. Se trata de un kit de un coste muy bajo que hace que sea ideal para nuestro propósito. La tarjeta se alimenta mediante USB por lo que no necesitamos ninguna fuente de alimentación externa salvo el propio puerto USB del PC.Este kit además incluye el IDE CodeComposer, del que se hablará más adelante, mediante el cual se programa y configura nuestro DSP. Todo esto abre la posibilidad de poder trabajar y experimentar con uno de los DSP de 16-bit de más bajo consumo que se puede encontrar en la actualidad: C5515.

Esta herramienta de bajo coste permitirá a los alumnos realizar una fácil y rápida evaluación de las avanzadas posibilidades que un procesador como el C5515 puede ofrecer.

Esta tarjeta confiere alDSP C5515 varios módulos periféricos como son el conector de expansión de40 pines, un puerto de expansión para un módulo bluetooth, 1 puerto USB 2.0,ranura para lector de tarjetas microSD, dos pulsadores, un conjunto de LEDs yun códec de audio programable de 32 Bits modelo TLV320AIC3204 de Texas Instruments quetiene en sus puertos de entrada/salida dos conectores estéreo de audio tipo mini Jack o TLR, a través de los cuales se introducen las señales del generador y se sacanlas señales una vez procesadas.

La tarjeta también dispone de un bus serie que permite la conexióncon un

equipo host para transmitir a través del emulador embebido del XDS100JTAG el programa de carga y depuración del DSP.

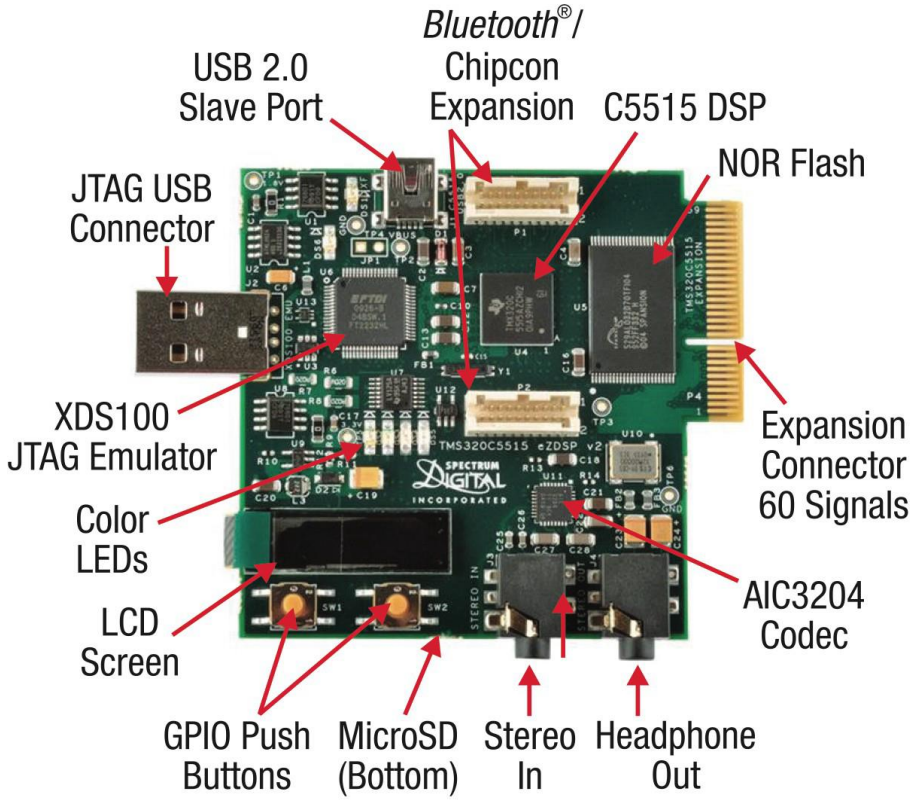

Figura 1. Detalle de la tarjeta de desarrollo TMS320C5515

En la próximas líneasse va a realizar una breve descripción delos dos elementos principales de nuestra tarjeta, el procesador DSP C5515 y el CODEC de audio TLV320AIC3204.

2.1.1 C5515

El C5515 es uno de los miembros de la familia de procesadores TMS320C5000 de punto fijo de Texas Instruments diseñado para realizar aplicaciones de bajo consumo.Los DSP de punto fijo están basados en la generación de procesadores TMS320C55x. La arquitectura de la familia de DSPs C55x logra un rendimiento alto pero manteniendo un bajo consumo. Esto se consigue aumentando el paralelismo computacional y prestando un cuidado especial en reducir el consumo.

La CPU da soporte a una estructura interna de buses que se compone por un bus de programa, un bus de lectura de datos de 32 bit, dos buses de lectura de

datos de 16 bit, dos buses de escritura de datos de 16 bit, y algunos buses adicionales dedicados a los periféricos y al DMA (este último encargado de la lectura y escritura de los datos del CODEC). Estos buses ofrecen la posibilidad de realizar hasta cuatro lecturas y dos escrituras de datos de 16 bit en un único ciclo. El dispositivo incluye además cuatro controladores DMA, cada uno con cuatro canales, proporcionando la posibilidad de mover datos por dieciséis canales independientes sin la necesidad de la actuación de la CPU. Cada controlador DMA puede realizar una transferencia de datos de 32 bit por ciclo, en paralelo y de manera independiente de la actividad de la CPU.

La CPU C55x dispone de dos unidades multiplicadoras-acumuladoras (MAC), cada una capaz de realizar multiplicaciones de 17 bit x 17 bit y una suma de 32 bit en un único ciclo. Dispone además de una unidad aritmético lógica (ALU) de 40 bits que se apoya en otra de 16 bit. El uso de las ALUs se realiza mediante un conjunto de instrucciones de control, que proporcionan la habilidad de realizar operaciones en paralelo con un bajo consumo de energía. Estos recursos son manejados por la unidad de direcciones (AU) y la unidad de datos (DU) de la CPU C55x.

La CPU C55x soporta un conjunto de instrucciones de ancho variable para poder optimizar la densidad del código. La unidad de instrucciones (IU) realiza desplazamientos de 32 bit de programa desde la memoria, interna o externa, y la cola de instrucciones hacia la unidad de programa (PU). La unidad de programa decodifica las instrucciones, dirige las tareas de la unidad de direcciones (AU) y los recursos de la unidad de datos (DU), y gestiona el *pipeline.*

La comunicación serie es soportada a través de dos periféricos *MultiMediaCard/Secure Digital* (MMC/SD), cuatro módulos Inter-IC Sound (I2S Bus), una interfaz Serial-Port (SPI) con selección de hasta 4 chips, una interfaz I2C y una interfaz Universal Asynchronous Receiver/Transmitter (UART).

El conjunto de periféricos del dispositivo incluye un interfaz de memoria externa (EMIF) que proporciona acceso sin colas a memorias de tipo asíncrono como las EPROM, NOR, NAND y SRAM así como a memorias de alta velocidad o alta densidad como las de tipo síncrono DRAM (SDRAM) y de tipo móvil SDRAM (mSDRAM). Además incluye algunos periféricos adicionales como un

puerto USB universal de alta velocidad (USB 2.0) y un reloj de tiempo real (RTC). Este dispositivo incluye también tres timers de propósito general, uno de ellos configurable como *timerwatchdog*, y un generador de reloj APLL (AsynchronousPhaseLockedLoop)

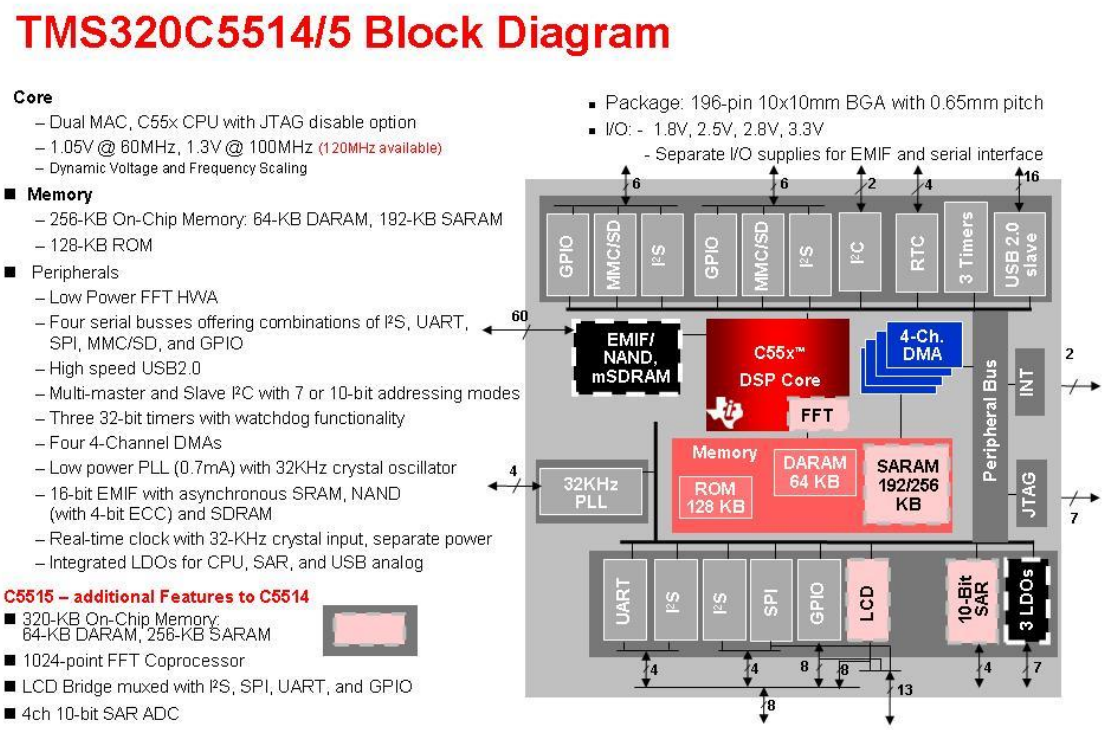

Figura 2. Diagrama de bloques del TMS320C5515

Adicionalmente incluye un acelerador hardware FFT que soporta valores reales y complejos desde 8 hasta 1024 puntos (en potencias de 2).

Por último, contiene tres LDOs integrados (DSP_LDO, ANA_LDO y USB_LDO) para proporcionar alimentación a diferentes secciones del dispositivo. El DSP_LDO proporciona 1.3V o 1.05V al core del DSP (CVDD), y puede ser seleccionado en marcha a través de software. Para conseguir un consumo menor de energía, el desarrollador puede desactivar el DSP_LDO interno cortando la energía al core del DSP (CVDD) mientras una fuente de alimentación externa proporciona la energía al RTC (CVDDRTC y DVDDRTC). El ANA_LDO está diseñado para suministrar 1.3V al PLL del DSP (VDDA_PLL), al SAR y los circuitos de administración de energía (VDDA_ANA). El USB_LDO suministra 1.3V al núcleo digital del USB (USB_VDD1P3) y a los circuitos de la capa física (USB_VDDA1P3). La interrupción de alarma RTC o el pin WAKEUP

pueden volver a habilitar el DSP_LDO interno y volver a proporcionar energía al núcleo del DSP.

Este procesador es soportado por múltiples entornos de desarrollo como eXpressDSP o CodeComposer Studio. Este último incluye herramientas para generación de código, incluyendo un compilador de C, drivers para los dispositivos de emulación RTDX, XDS100, XDS510 y XDS560, y algunos módulos de evaluación. Puede ser utilizado con la librería C55x DSP que cuenta con más de 50kernels software (filtros FIR, filtros IIR, FFTs y múltiples funciones matemáticas) así como librerías soportadas por el chip.

2.1.2 TLV320AIC3204

El TLV320AIC3204 (a veces referido como el AIC3204) es un códec de audio estéreo de bajo consumo y que requiere tensiones bajas de alimentación. Tiene entradas y salidas programables, tecnología PowerTune, bloques de procesamiento de señales fijos, predefinidos y parametrizables, un PLL integrado, LDOs integrados e interfaces digitales flexibles.

El códec incluye un registro extensivo que se encarga del control del consumo, la configuración de los canales de entrada salida, las ganancias, los efectos, la multiplexación de pines y los relojes, permitiendo al dispositivo ser preciso en la ejecución de sus aplicaciones. Combinado con la avanzada tecnología PowerTune, el dispositivo puede realizar operaciones, desde reproducciones de voz mono a 8Khz hasta reproducciones de audio estéreo a 192Khz, haciéndolo ideal para ser utilizado en reproductores portátiles de audio o para aplicaciones de telefonía.

La ruta de grabación del TLV320AIC3204 puede realizar grabaciones de audio desde mono a 8 KHz hasta estéreo a 192 KHz, y contiene las configuraciones programables del canal de entrada que pueden ser de tipo diferencial o de tipo single-ended, así como señales de entrada flotantes o mezcladas. También incluye un preamplificador estéreo de micrófono, controlado digitalmente y un bias de micrófono integrado. Contiene además bloques de procesado de señales digitales, capaces de eliminar el ruido causado por un acoplamiento

mecánico como puede ser el zoom óptico de una cámara digital.

La ruta de reproducción ofrece bloques de procesamiento de señales con los que filtrar y generar efectos. Soporta la mezcla de señales de entrada analógicas y DAC. Además tiene controles de volumen programables.El path de reproducción contiene dos controladores de salida de alta potencia, así como dos salidas totalmente diferenciales. Las salidas de alta potencia se pueden configurar de múltiples maneras, incluyendo estéreo y mono BTL.

La tecnología integrada PowerTune permite al dispositivo ser ajustado para un consumo mínimo sin sacrificar por ello su rendimiento. Lasaplicaciones móviles tienen múltiples casos de uso en entornos móviles donde es fundamental un bajo consumo de energía. Por el contrario cuando el dispositivo se utiliza mediante una fuente de energía externa el consumo del dispositivo adquiere un papel secundario, mientras que minimizar el ruido si es importante. Con PowerTune el TLV320AIC3204 puede ser utilizado en ambas situaciones.

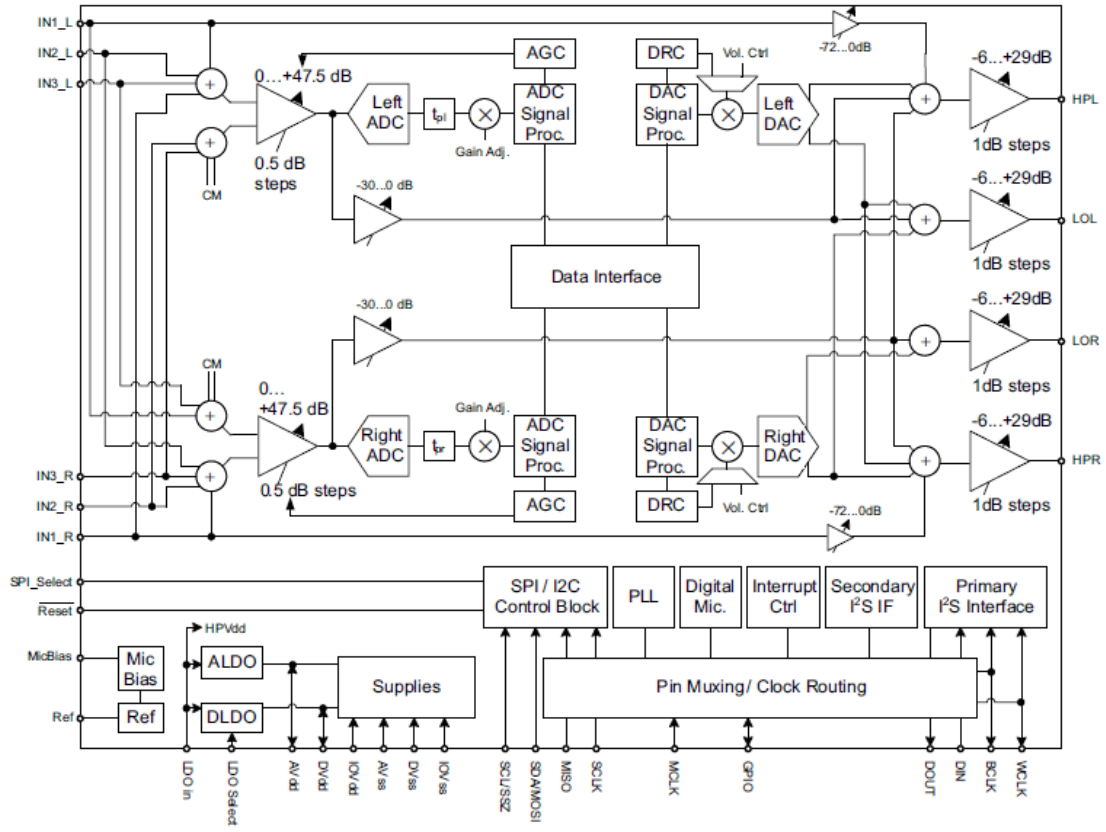

Figura 3: Diagrama de bloques simplificado del TLV320AIC3204

Los rangos de alimentación para el TLV320AIC3204 son de 1.5V a 1.95V para analógico, y de 1.26V a 1.95V para digital. Para facilitar el diseño a nivel de sistema, el códec integra una serie de LDOsque son utilizados para generar una apropiada alimentación tanto analógica como digital partiendo de un rango de tensiones de entre 1.8V y 3.6V. Para las entradas-salidas digitales son soportadas tensiones de 1.1V a 3.6V.

El reloj interno que requiere el TLV320AIC3204 puede derivarse de varias fuentes, incluyendo el pin MCLK, el pin BCLK, el pin GPIO o la salida del PLL interno, donde la entrada del PLL puede derivar de nuevo del pin MCLK, o de los pines BCLK o GPIO. Aunque utilizando el PLL aseguramos la disponibilidad de una señal de reloj adecuada, no es recomendado para configuraciones de bajo consumo. El PLL es altamente programable pudiendo aceptar señales de reloj de entrada desde los 512Khz hasta los 50Mhz. Es importante un correcto uso del PLL ya que será uno de los elementos utilizados para generar que las frecuencias de muestreo que van a requerir los desarrollos implementados durante las prácticas.

Con toda la información anteriormente expuesta daríamos por concluida la descripción del hardware empleado. Para llevar a cabo un estudio más en profundidad recomendamos dirigirse al website de Texas Instruments donde es posible encontrar una gran cantidad de documentación y datasheets que hacen referencia a los distintos elementos de hardware que hemos utilizado.

2.2 Software empleado.

Para poder completar todas las tareas del proyecto, se necesita utilizar tres programas: Matlab, CodeComposer Studio v.5.0 y SoundcardScope. A continuación se verá cada uno de ellos de una manera un poco más detallada, aunque sin llegar a profundizar, ya que no es uno de los objetivos perseguidos en este proyecto.

2.2.1 SoundcardScope

Se trata de un software que basa su funcionamiento en el uso de la tarjeta de sonido del PC como interfazdeentrada/salida. Es un software de libre distribución siempre que su uso vaya orientado a fines educativos.

Recibe los datos desde la tarjeta de sonido con una frecuencia de 44.1 KHz y una resolución de 16 bits, pudiéndose seleccionar el origen de los datos desde el mezclador de Windows. El rango de frecuencias va a depender de la tarjeta de sonido empleada, un rango de 20Hz a 20KHz no debería suponer un problema con cualquier tarjeta moderna.

Este programa presenta varias funcionalidades que son de gran utilidad, siendo la principal de ellas la de osciloscopio. Además de esta, también ofrece un generador de señales, un analizador de espectros y un módulo de representación en modo X/Y con el que poder generar figuras de Lissajous.

La función de osciloscopio contiene todas las funcionalidades necesarias para poder realizar las medidas y comprobaciones que serán necesarias durante el desarrollo de las prácticas.Se pueden realizar ajustes tanto en amplitud como en tiempo para poder ver representadas las señales de la forma más adecuada, pudiendo realizar medidas tanto de tensión como de frecuencia, o mediante cursores para hacer medidas personalizadas. Incluye cuatro modos de disparo y además incluye la posibilidad de poder configurarlo con el ratón para fijar el punto de disparo donde resulte más interesante.A nivel de representaciónpermite representar dos canales de forma simultánea o individual, pudiendo sumarlos, restarlos o multiplicarlos.

También existe la posibilidad de representar las señales de entrada en modo X/Y que puede utilizarse para deducir el desfase entre dos señales y su relación de frecuencias. Esta función solo se podrá emplear cuando se trate de funciones senoidales.

Otra de las funcionalidades que de gran utilidad es el generador de señales, se trata de un generador de dos canales que puede generar ondas senoidales, cuadradas, triangulares y de diente de sierra en un rango de frecuencia de 20Hz a 20KHz y una amplitud máxima de 1V. Las señales generadas se entregan en la salida audio de la tarjeta de sonido.

Por último dispone de un módulo para la representación del espectro en frecuencia de la señal, con algunas funciones, como la retención de pico, que puede ser de gran utilidad por ejemplo, para a la hora de comprobar a que frecuencia está muestreando el códec o a la hora de ver el comportamiento de

los filtros implementados.

2.2.2 CodeComposer Studio

Para programar el DSP se utiliza el lenguaje de programación C, para tal fines necesario hacer uso de un entorno de desarrollo. En el kit elegido para este trabajo se incluye CodeComposer Studio (CCS), que es un software propiedad de Texas Instruments.

CodeComposer Studio es un entorno de desarrollo integrado (IDE) que soporta toda la familia de microcontroladores y procesadores integrados de Texas Instruments. CodeComposer Studio incluye un conjunto de herramientas para desarrollar y depurar aplicaciones embebidas. Incluye un compilador de C/C++ optimizado, un editor de código fuente, un entorno de compilación de proyectos, un depurador, y muchas otras funcionalidades. Se trata de un IDE muy intuitivo que proporciona una única interfaz que llevará al usuario a través de cada paso del proceso del desarrollo de las aplicaciones. Incluye interfaces y herramientas que ya son familiares para el usuario lo que le permite empezar a trabajar más rápido que nunca. CodeComposer Studio combina las ventajas de la plataforma de software Eclipse con las avanzadas habilidades de depuración de aplicaciones embebidas de Texas Instruments, dando como resultado un entorno de desarrollo rico en posibilidades para desarrolladores de sistemas integrados.

De todas las funcionalidades que integra, la que más se ha utilizado a lo largo del proyecto ha sido el depurador, por su facilidad de uso y porque permite gran cantidad de opciones, como leer variables o hacer representaciones gráficas para ver si las señales se están procesando de forma correcta. Otra de las posibilidadesque ofrece eldepurador, espoder pausar la ejecución del programa para poder realizar comprobaciones del contenido de las variables y registros,yseguir ejecutando paso a paso.o volver a iniciar su ejecución sin necesidad de volver a compilar el proyecto.

CCS es sin duda una gran herramienta por su versatilidad y por su facilidad de uso.

2.2.3 Matlab

Para poder realizar correctamente la parte referente al filtrado digital de este proyecto, es necesario contar con una herramienta que permita calcular y diseñar filtros de manera precisa y fiable. El cálculo de los coeficientes de un filtro como puede ser un FIR o un IIR es algo que puede convertirse en una tarea muy tediosa, pero conla ayuda de un programa adecuado puede convertirse en una tarea rápida y sencilla, es por todo esto que la herramienta elegida no podía ser otra que Matlab, que no solo permite calcular los filtros, sino que además sirve para verificar si los resultados obtenidos son los correctos.

Matlab (MATrixLABoratory) es un programa orientado al cálculo con vectores y matrices a lo que se reducen muchos de los algoritmos con los que son resueltos gran parte de los problemas de matemática aplicada o ingeniería. Lo que distingue a Matlab de otros programas, es precisamente su facilidad a la hora de trabajar con vectores y matrices. Las operaciones más frecuentes, suma, producto, potencia, operan por defecto sobre matrices, con la única restricción de la compatibilidad de tamaños en cada caso.

Ofrece un entorno interactivo sencillo mediante una ventana en la que se pueden introducir comandos en modo texto y en la que se visualizan los resultados. Los resultados gráficos se ven en ventanas independientes de la ventana principal, disponiendo en cada ventana de una barra de menús para el manejo de las distintas funcionalidades.

Además Matlab incluye una serie de módulos clasificados por materia que ofrecen funcionalidades especificas.Para este proyecto se ha hecho uso de la toolbox de procesado de señales que incluye, entre otras, la herramienta Fdatool (FilterDesign and AnalysisTool).

Fdatool ha sido la herramienta elegida para diseñar los filtros que se utilizan durante el desarrollo de las prácticas de filtrado.Los motivos fundamentales por los que se ha decidido hacer uso de esta herramienta son principalmente dos, el primero por su facilidad de uso, ya que tiene una interfaz muy intuitiva y de fácil comprensión, y el segundo porque a la hora de diseñar los filtros permite exportar los coeficientes del filtro a un fichero de cabecera ".h" que se podrá

incluir en el proyecto de CCS de forma muy sencilla.

A la hora de diseñar un filtro,Fdatool ofrece infinidad de parámetros configurables como el tipo de respuesta, el algoritmo de diseño, el orden del filtro y las diferentes especificaciones a nivel de frecuencia o magnitud, entre otras. Una vez diseñado el filtro se pueden conocer todas sus características como pueden ser el diagrama de polos y ceros, la respuesta en frecuencia, la respuesta en magnitud, los coeficientes y algunas otras. Algunas de estas características son presentadas de forma gráfica de manera que será muy fácil verificar si el filtro calculado va a tener el comportamiento deseado a la hora de realizar el diseño.

3 Diseño e implementación de las unidades didácticas.

Como ya se ha visto en la introducción, el objetivo final es diseñar cinco unidades didácticas o prácticas, mediante las cuales el alumno pueda reforzar sus conocimientos sobre el procesado digital de señales.

En cada una de las prácticas se hará correr un programa en el DSP. Para poder realizar las diferentes implementaciones se ha partido de un proyecto de ejemplo ya generado que únicamente lee los datos de la entrada del ADC y los pasa a la salida de DAC. Sobre este proyecto se harán las modificaciones necesarias para adaptarlo a cada una de las prácticas.

En esta sección se explican todos los pasos seguidos para llevar a cabo la implementación y definición de dichas unidades didácticas, analizando cada una de ellas por separado.

3.1 Práctica 1: El entorno de desarrollo.

Los objetivos perseguidos en esta primera práctica son los siguientes:

- Familiarizarse con la tarjeta de desarrollo TMS320C5515, Scope y el CodeComposer Studio (CCS).
- Conocer las recomendaciones básicas y el procedimiento inicial para utilizar la tarjeta.
- Compilar y depurar un proyecto en CCS y cargarlo en la tarjeta.
- Realizar los ajustes en la configuración de audio de Windows para aprovechar el margen dinámico del códec.

Para alcanzar estos objetivos, después de una breve descripción del hardware, se han planteado una serie de cuestiones acerca del DSP y el CODEC que el alumno deberá completar con ayuda de los datasheets del fabricante. De esta manera, el alumno tendrá unos mínimos conocimientos del hardware que se va a emplear a lo largo de todo el laboratorio.

Seguidamente se realiza una pequeña práctica guiada, que hace la función de tutorial, en la que el objetivo es que el alumno se familiarice con el entorno de trabajo. En ella se explica todo el proceso que hay que seguir para la integración de la tarjeta en el entorno de trabajo y su puesta en marcha.

En esta guía se describe cómo conectar la tarjeta al PCy cómo cargar un proyecto en CodeComposer, con una breve descripción de los tipos de archivos de los que se compone un proyecto de CCS. Una vez que cargado el proyecto, se explican los pasos a seguir para compilar y depurar un programa, con una pequeña guía de uso de las diferentes funcionalidades del depurador que serán de gran ayuda a la hora de comprobar resultados o de detectar errores en el programa. Además se explica cómo poner en marcha la herramientaScope para generar las señales de entrada a nuestra tarjeta y poder visualizar la salida de la misma.

Es en este momento, una vez que el programa está ejecutándose en el DSP, cuando se pedirá al alumno que realice los ajustes necesarios sobre la configuración de sonido de Windows para poder aprovechar al máximo el margen dinámico del códec, de manera que en todo momento la salida corresponda con lo que se configura en el generador de señales de Scope.

Por último, una vez realizados los ajustes de la tarjeta de sonido, el alumno deberá completar una serie de cuestiones, con las queexperimentará algunas de las funcionalidades tanto de CCS como del Scope.

3.2 Práctica 2: Muestreo

Sin duda la teoría de muestreo es el pilar principal sobre el que está construida toda la teoría de procesado digital de señales, es por ello que se ha dedicado una práctica en exclusiva.

Los objetivos que se quieren conseguir, son los siguientes:

- Conocer y manipular los parámetros básicos de configuración del códec.
- Analizar cuáles son los efectos del muestreo sobre la señal.
- Analizar el efecto del submuestreo.

- Manejar cada canal (R y L) independientemente.

Al finalizar la práctica dos, el alumno tiene que ser capaz de configurar la frecuencia de muestreo del códec en función de las necesidades y los requerimientos del sistema.

Durante el desarrollo de esta práctica se realizan modificaciones sobre la señal de entrada que seescribirá a la salida para poder visualizarla con Scope.

Llegado a este punto es necesario explicar algunos detalles del funcionamiento del códec, como del funcionamiento del programa que se usará a lo largo del laboratorio.

El proyecto CCS del que se ha partido incluye una serie de librerías ya codificadas, a partir de las cuales se han implementado todos los programas. En el caso de este proyecto la más interesante es la librería con las funciones propias del códec (BSL_aic3204.c) ya que son algunas de estas funciones, a las únicas que se llama desde el programa principal. En concreto van a ser cuatro funciones:

- voidBSL_AIC3204_init( ): Esta es la función que inicializa nuestro códec. Esta función es la encargada de escribir todos los registros de configuración del códec con los parámetros que se hayan definido. Para este proyecto, los únicos registros que van a modificar son aquellos que configuran la frecuencia de muestreo. Además dentro de esta función se van a inicializar también el I2C, I2S y el DMA.
- voidBSL_AIC3204_start( ): Esta función es la que arranca el códec, para ello lo que hace es habilitar el I2S y el DMA.
- void BSL_AIC3204_read (structstereo *data): Esta función es con la que se van a leer los datos del ADC, se le pasa como parámetro un puntero a una estructura del tipo estero donde se escribirán los valores de entrada tanto del canal derecho como del canal izquierdo.
  La estructura de tipo estéreo se ha definido de la siguiente manera:

```
#define BUFFER_SIZE 128
struct stereo {
      Int16L[BUFFER_SIZE];
      Int16R[BUFFER_SIZE];
};
```

Se trata de una estructura en la que se almacenan dos buffer de tipo Int16 (entero de 16 bits) y tamaño 128. "L" hace referencia al canal izquierdo (Left) y "R" hace referencia al canal derecho (Right). Este es un dato a tener en cuenta, ya que a la hora de escribir en los buffer de salida es importante no confundirlos.

- void BSL_AIC3204_write (structstereo *data): Esta función trabaja igual que la de lectura, solo que en este caso se trata de la función que escribe los datos de salida.

Como se ha comentado en líneas anteriores el alumno debe ser capaz de configurar la frecuencia de muestro tanto del ADC como del DAC delcódec. Para ello es necesario modificar una serie de registros que se configuran dentro de la función voidBSL_AIC3204_init( ). En las próximas líneas se verá cómo calcular la frecuencia de muestreo y cuáles son las líneas de código que se deben modificar.

Haciendo uso del datasheet del códecse encuentran una serie de apartados, en los que se explica cómo configurar la F_s. En resumen la F_s va a depender de como este configurado el reloj del sistema, en este caso, haciendo uso del PLL que integra el códec y de los valores configurados en una serie de registros que se van a ir viendo.

Lo primero que hay que saber es la fuente de reloj que se está utilizando, ya que la F_s tanto del ADC como del DAC, puede generarse a partir de distintas fuentes de reloj, como se puede ver en la figura 4. En este caso se utiliza el MCLK (Master Clock) y a partir de ahí se realizan una serie de operaciones para conseguir la F_s deseada.

Las fórmulas a través de las se va a configurar el DAC y el ADC, son las siguientes:

$$ADC\_FS = \frac{CODEC\_CLKIN}{NADCxMADCxAOSR}$$

$$DAC\_FS = \frac{CODEC\_CLKIN}{NDACxMDACxDOSR}$$

Donde NADC, MADC, AOSR, son los registros que configuran la F_s del ADC y NDAC, MDAC, DOSR los registros que configuran el DAC.

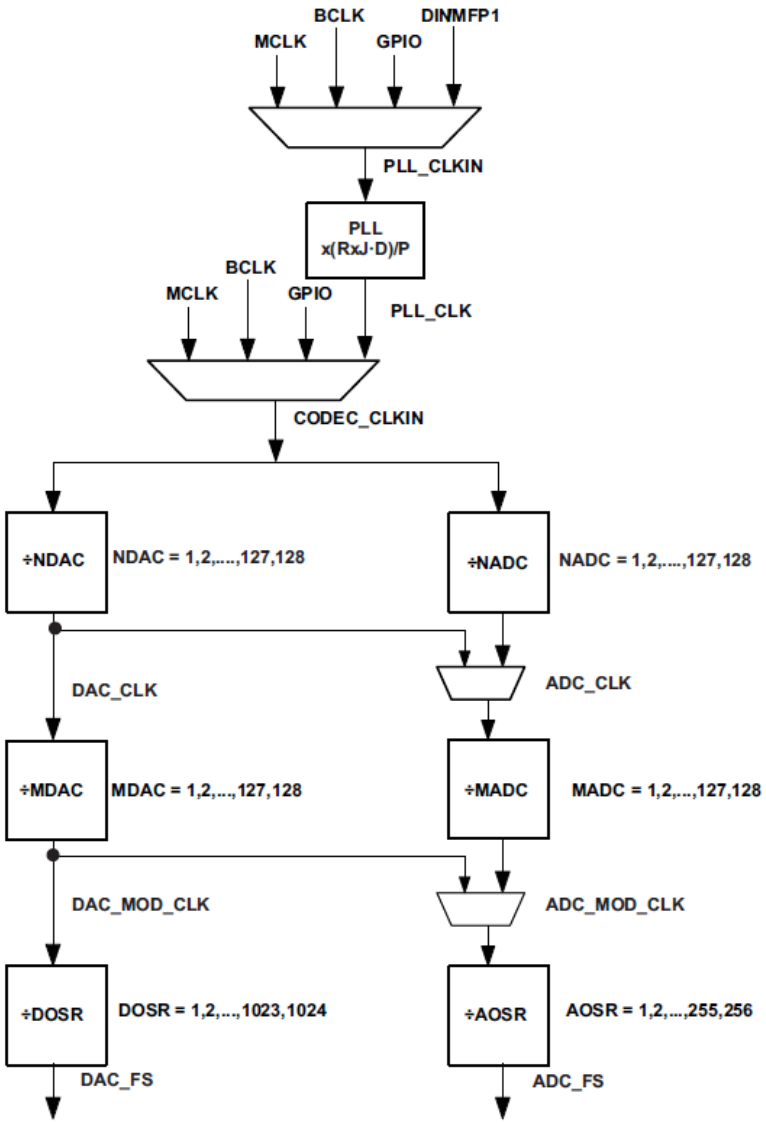

Figura 4: Árbol de distribución de la fuente de reloj.

CODEC_CLKIN es la señal de reloj seleccionada y que también se puede configurar con una serie de registros, ya que en este caso se ha utilizado el PLL como fuente de reloj.

$$PLL\_CLK = \frac{PLL\_CLKIN x RxJ.D}{P}$$

Donde R, J, D y P son registros configurables y que toman los siguientes valores:

- R = 1, 2, 3,4
- J = 4,5,6,... 63, y D = 0, 1, 2... 9999
- P = 1, 2, 3... 8

Una vez visto esto, Texas Instruments ofrece unas tablas donde se pueden ver diferentes combinaciones de todos estos parámetros para conseguir unos valores de F_s

Fs = 44.1kHz										
MCLK (MHz)	PLLP	PLLR	PLLJ	PLLD	MADC	NADC	AOSR	MDAC	NDAC	DOSR
2.8224	1	3	10	0	3	5	128	3	5	128
5.6448	1	3	5	0	3	5	128	3	5	128
12	1	1	7	560	3	5	128	3	5	128
13	1	2	4	2336	13	3	64	4	6	104
16	1	1	5	2920	3	5	128	3	5	128
19.2	1	1	4	4100	3	5	128	3	5	128
48	4	1	7	560	3	5	128	3	5	128
Fs = 48kHz										
2.048	1	3	14	0	2	7	128	7	2	128
3.072	1	4	7	0	2	7	128	7	2	128
4.096	1	3	7	0	2	7	128	7	2	128
6.144	1	2	7	0	2	7	128	7	2	128
8.192	1	4	3	0	2	8	128	4	4	128
12	1	1	7	1680	2	7	128	7	2	128
16	1	1	5	3760	2	7	128	7	2	128
19.2	1	1	4	4800	2	7	128	7	2	128
48	4	1	7	1680	2	7	128	7	2	128

Figura 5: Ejemplos de configuración del PLL.

Una vez que se ha explicado cómo se pueden configurar las frecuencias de muestreo tanto del DAC como del ADC, solo faltaría ver cuáles son los registros que hay que modificar para conseguir las frecuencias deseadas. En concreto las líneas de código que será necesario modificar son las siguientes:

Para configuración del CODEC_CLKIN:

```
// CODEC_CLKIN Configuration (w/ PLL)
BSL_AIC3204_rset(  0, 0x00 );   // Select page 0
BSL_AIC3204_rset(  4, 0x03 );   // MCLK -> PLL -> CODEC_CLKIN
BSL_AIC3204_rset(  6, 0x07 );   // J = 7
BSL_AIC3204_rset(  7, 0x06 );   // D - HI_BYTE = 1680 (0x0690)
BSL_AIC3204_rset(  8, 0x90 );   // D - LO_BYTE = 1680 (0x0690)
BSL_AIC3204_rset(  5, 0x91 );   // P = 1 | R = 1 | Power up PLL
```

Para configuración de las frecuencias de muestreo del ADC y el DAC:

```
// ADC_FS & DAC_FS Configuration
BSL_AIC3204_rset(  0, 0x00 );   // Select page 0
BSL_AIC3204_rset( 13, 0x00 );   // DOSR - Hi_Byte = 128 (0x0080)
BSL_AIC3204_rset( 14, 0x80 );   // DOSR - Lo_Byte = 128 (0x0080)
BSL_AIC3204_rset( 20, 0x80 );   // AOSR = 128 (0x0080) - Filter A
BSL_AIC3204_rset( 11, 0x82 );   // Power up and set NDAC value to 2
BSL_AIC3204_rset( 12, 0x87 );   // Power up and set MDAC value to 7
BSL_AIC3204_rset( 18, 0x82 );   // Power up and set NADC value to 2
BSL_AIC3204_rset( 19, 0x87 ); // Power up and set MADC value to 7
```

Estas líneasde código, como ya se ha comentado, se encuentran dentro de la función voidBSL_AIC3204_init( )

Ahora que ya se haaclarado cómo se debe calcular y configurar la F_s del ADC y el DAC, hay que ver cómo va a ser en sí el desarrollo de la práctica.

Para alcanzar los conocimientos deseados, se han planteado ocho cuestiones que el alumno deberá ir resolviendo:

- Las dos primeras cuestiones están más centradas en que el alumno realice una serie de operaciones con Scope para que se familiarice con algunas de sus características.
- La tercera cuestión es la primera en la que el alumno deberá poner en práctica sus conocimientos sobre la teoría de muestreo. En concreto el alumno deberá deducir la frecuencia de muestreo a la que está configurada nuestro códec a la vista de la respuesta en frecuencia de la señal.
- La cuarta, quinta y sexta, son diferentes cuestiones en relación a la frecuencia de muestreo y de cómo configurarla, modificando los diferentes registros.
- Por último en las cuestiones siete y ocho se ha pedido al alumno que modifique la señal de entrada (una sinusoide) para obtener a la salida una señal cuadrada y un diente de sierra respectivamente.

Con todo esto, el alumno debería alcanzar los objetivos marcados al principio de la práctica.

3.3 Práctica 3: Filtrado FIR

El filtrado digital es una de las partes más importantes del procesado digital de señales, de ahí que tanto esta práctica 3, como la 4, se han dedicado ala implementación de filtros, en concreto en esta práctica a filtros de tipo FIR.

Los objetivos que el alumno debería alcanzar al finalizar esta práctica son los siguientes:

- Diseñar y caracterizar un filtro FIR con ayuda de Matlab, incluyendo el cálculo de los coeficientes.
- Exportar desde Matlab los coeficientes a CCS para su implementación.
- Interpretar los resultados obtenidos una vez que haya conseguido implementar el filtro diseñado en el DSP.

En esta ocasión se trata de guiar al alumno a lo largo de la práctica 3 para que una vez finalizada, haya alcanzado los objetivos marcados.

El desarrollo de esta práctica se ha realizado en varios paso que a continuación se detallan.

Para poder implementar un filtro FIR en CCS a partir de sus coeficientes, lo primero que se ha hecho, ha sido estudiar la manera más sencilla de calcular los coeficientes para evitar complicaciones derivadas de un mal uso de Matlab. La herramienta Matlab ofrece múltiples posibilidades para calcular un filtro FIR y sus coeficientes, pero hay una que sin duda es la más apropiada para el alumno; es el caso de la herramienta Fdatool que forma parte del modulo de procesado de señales. Con esta herramienta el alumno debe ser capaz de calcular un filtro FIR sin complicaciones.

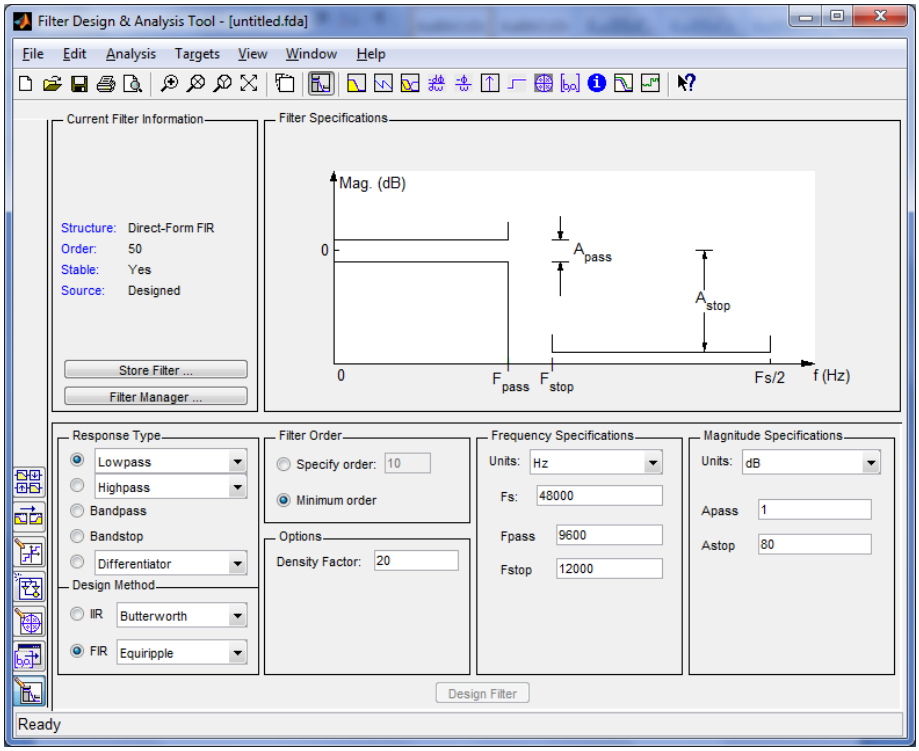

Figura 6: Interfaz gráfica de Fdatool.

Como se puede ver en la imagen simplemente es necesario rellenar una serie de campos que la herramienta propone y pulsar el botón "DesignFilter" y se mostrará el resultado del filtro calculado, como se puede ver en la siguiente imagen.

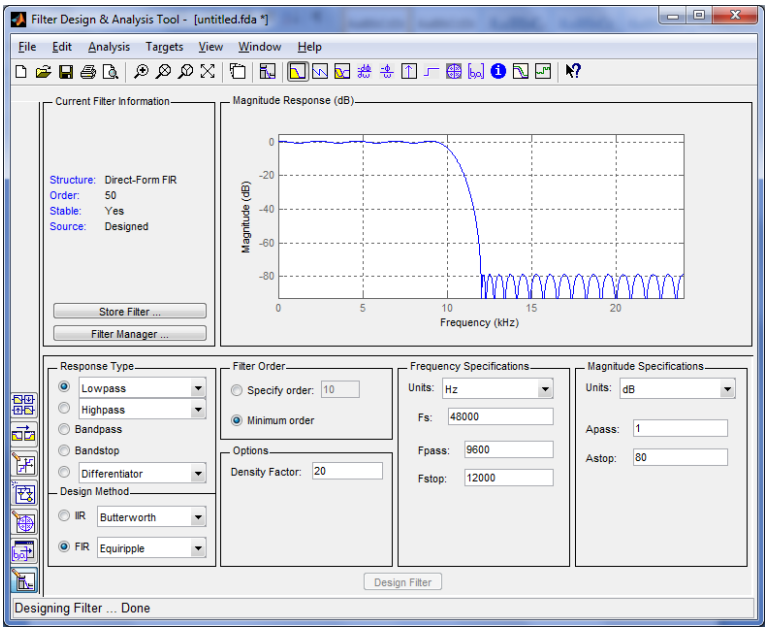

Figura 7: Respuesta en magnitud de un filtro calculado.

Una consideración a tener en cuenta a la hora de configurar los diferentes parámetros es que la F_s elegida debe coincidir con la configurada en el DSP.

Una vez diseñado el filtro elegido, es el momento de exportar los coeficientes. La herramienta Fdatoolfrece la posibilidad de generar un fichero de tipo header (".h").

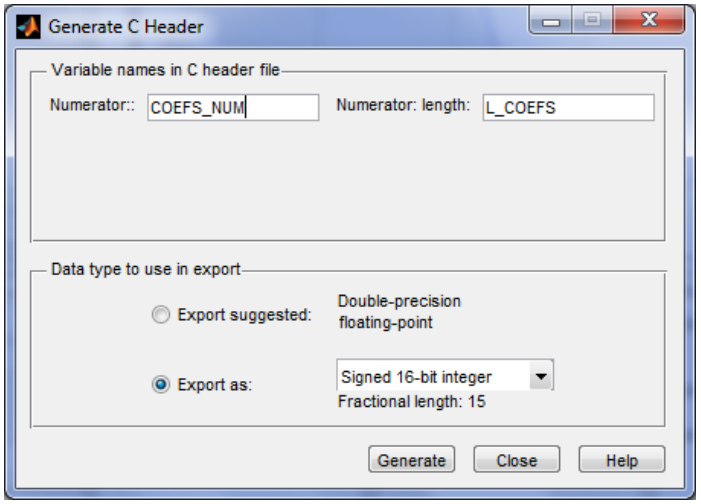

Figura 8: Ventana de exportación de los coeficientes del filtro.

Llegado este punto hay que tener en cuenta otro de los puntos importantes en el desarrollo de este proyecto. Como ya se ha dicho, el C5515 es un DSP en punto fijo, esto implica que a la hora de exportar los coeficientes es necesario cuantificarlos, ya que Matlab hace el cálculo de los coeficientes en aritmética de punto flotante. Esta tarea de cuantificar los coeficientes también la hace Fdatool, y a la hora de exportarlos se puede seleccionar el tipo de dato con el que se exportan, en este caso, entero de 16 bits con signo. En la siguiente práctica, se verán algunos de los problemas que la cuantificación de coeficientes puede acarrear.

Ahora que el filtro ya está diseñado y generado el fichero de coeficientes, el alumno solo tendría que incluirlo en el proyecto con el resto de ficheros de cabecera y ejecutar el programa para verificar su funcionamiento. Para que esto sea así, ha sido necesario codificar en C el algoritmo que implementa un filtro FIR a partir de sus coeficientes, a continuación se explica cómo se ha hecho.

Para programar en C el filtro FIR, se ha partido de la siguiente ecuación:

$$y(n) = \sum_{l=0}^{L-1} b_l x(n-l)$$

Donde *L* es el orden del filtro y b_l los coeficientes del filtro.

Lo primero que hemos hecho ha sido definir una función de tipo "Int16" que implementa la el filtrado FIR y devuelve el valor de y(n).

```
Int16fir_convolution (Int16 *x, constInt16 *h, intntap)
{
	Int32yn = 0;			/* Output of FIR filter */
	inti;				/* Loop index */

	for(i=0; i<ntap; i++)
	{
	yn += (Int32)h[i]*(Int32)x[i];/* Convolution of x(n) with h(n) */
	}

	yn = yn>> 15;
	return((Int16)yn);
}
```

En el cuadro se puede ver la parte del código donde se implementa la función "fir_convolution". Los parámetros de la función son:

- *x: puntero al array donde se almacenan las muestras de la señal a filtrar.
- *h: puntero al array de coeficientes del filtro.
- ntap: es el orden del filtro.

Cuando se multiplican datos de 16 bits, el resultado puede dar un dato de 32 bits, es por eso que se ha definido "yn" como "Int32". Y por este motivo ha sido necesario realizar un casting a las variables "h" y "x", en la operación de convolución.

Una vez finalizado el sumatorio se obtiene el resultado en 32 bits (yn), por tanto, para evitar problemas de overflow, es necesario desplazar el resultado 15 bits hacia la derecha. Ya solo faltaría devolver el resultado, no sin antes haber convertido en "Int16" la variable "yn", de ahí que se realice el casting en la línea donde se hace el "return".

Con esto ya estaría completa la función que realiza el filtrado FIR, pero según esta codificada, la función va a filtrar tantas muestras de la señal como coeficientes tenga el filtro. En este desarrollo se trabaja con un buffer de la señal de entrada de 128 muestras, por ese motivo si el filtro tiene menos de 128 coeficientes se estarían dejando muestras sin filtrar de la señal de entrada. Para solucionar este problema se ha definido otra función que lo que hace básicamente es desplazar las muestras, hasta que se hayan pasado por el filtro las 128.

```
voidshift (Int16 *x, intntap)
{
	inti;                                   /* Loop index */
	for(i=ntap-1; i> 0; i--)
	{
		x[i] = x[i-1];                  /* Shift old data x(n-i) */
	}

}
```

Como se observa, esta función lo único que hace es desplazar todos los datos una posición a la derecha, dejando libre la primera posición, que será donde se introduzca el nuevo dato de nuestra señal de entrada.

Para concluir con la programación del filtro, se han integrado las dos funciones anteriores en una única función que realiza un filtrado de M muestras, en este caso siempre será 128 que es el tamaño que tienen el array que entrega la función que lee los datos del ADC.

```
voidfir (Int16 *in, int M, constInt16 *h, intntap, Int16 *out, Int16 *x)
{
	int j;			// Loop index
	for(j=0; j < M; j++)
	{
		x[0] = in[j];		// Insert new data x(n)

		out[j] = fir_convolution(x, h, ntap);
		// store convolution result to output buffer

		shift(x, ntap);		// Shift old data x(n-i)
	}
	return;
}
```

Los parámetros de la función son:

- *in: puntero al buffer de la señal de entrada
- M: es el número de muestras que van a filtrar, en este caso 128.
- *h: puntero al array de coeficientes
- ntap: es el número de coeficientes del filtro
- *out: puntero al array donde se almacenan los datos procesados que posteriormente serán escritos a la salida.
- *x: puntero al buffer donde vamos a almacenar parte de las muestras de la señal de entrada, y que conforme se vayan filtrando se irán desplazando para hacer que todas las muestras pasen por el filtro.

Ahora sí, estaría todo lo necesario para ejecutar el filtro, de forma que el programa principal quedaría de la siguiente manera:

```
voidmain ( )
{
	struct stereo dataIn;
	struct stereo dataOut;

	//Reserve memory
	Int16 *signal_buffer=(Int16 *)calloc(BL, sizeof(Int16));

	BSL_AIC3204_init( );// Initialize CODEC
	BSL_AIC3204_start( );// Start CODEC

	while(1) {// Enter the Main Loop

	BSL_AIC3204_read(&dataIn); // Read 16-bit stereo data

	//block FIR to right channel
	fir(dataIn.R, BUFFER_SIZE, COEF, BL, dataOut.R, signal_buffer);

	BSL_AIC3204_write(&dataOut);// Write 16-bit stereo data
	}
}
```

Tal y como se ve, el filtrado solo se realiza sobre el canal derecho, si se quisiese se podría añadir otra línea y filtrar con otro filtro distinto, sobre el canal izquierdo.

Con esto, ya estaría todo listo para poder ejecutar el filtro diseñado con Matlab y ver los resultados obtenidos en Scope. Para ver cómo se comporta el filtro, bastaría con configurar el generador de señal en modo barrido de frecuencia, y observar con el analizador de espectro en frecuencia, como se comporta nuestra señal.

Para que el alumno alcance los objetivos marcados, se le pedirá que diseñe una serie de filtros (paso bajo, paso alto y paso banda) y los implemente en el DSP. Una vez implementado deberá ser capaz de evaluar el resultado obtenido a partir de las especificaciones originales. Deberá ser capaz de analizar los problemas que podría acarrear la implementación realizada del filtro, como pueden ser el overflow o la perdida de precisión si tenemos más de 128 coeficientes. Por otro lado también debe ser capaz de definir el margen dinámico de nuestro filtro y ver si tiene alguna limitación en cuanto a la tensión de la señal de entrada.

Una vez superado esto, el alumno habría alcanzado los objetivos marcados al principio del práctica.

3.4 Práctica 4: Filtrado IIR

Si en la práctica 3 seveía la implementación de los filtros FIR en este práctica es el turno del filtrado IIR. Los objetivos que el alumno deberá alcanzar son muy parecidos a la anterior, ya que se trata de un desarrollo equivalente.

Los objetivos que el alumno debería alcanzar al finalizar esta práctica son los siguientes:

- Diseñar y caracterizar un filtro IIR con ayuda de Matlab, incluyendo el cálculo de los coeficientes.
- Exportar desde Matlab los coeficientes a CCS para su implementación.
- Interpretar los resultados obtenidos una vez que haya conseguido implementar el filtro diseñado en el DSP.
- Hacer un análisis de ventajas e inconvenientes de la implementación de un filtro IIR frente a uno FIR.

Como la primera parte de la práctica es igual que la anterior se va a comentar directamente cómo ha sido programado el filtro ya que la parte del diseño y la exportación del filtro se hace exactamente igual. Simplemente mencionar, que a la hora de cuantificar los coeficientes esta vez el resultado de la cuantificación no es tan bueno como en el caso de los FIR y será tarea del alumno analizar si el resultado obtenido es válido.

Un filtro IIR se puede caracterizar por la siguiente ecuación en diferencias:

$$y(n) = \sum_{l=0}^{L-1} b_l x(n-l) - \sum_{m=1}^{M} a_m y(n-m)$$

Donde:

- L: es el número de coeficientes *b*
- M: es el número de coeficientes *a*
- *b* y *a*: son los coeficientes del filtro

Esa es la ecuación a partir de la cual se ha programado nuestro filtro con un desarrollo muy similar al del filtro FIR ya que como se puede observar, si los coeficientes *a* fuesen todos igual a cero, nuestro filtro IIR tendría la misma

forma que un filtro FIR, de ahí que a la hora de programarlo haya resultado muy sencillo.

Para programar el filtro simplemente se han programado los dos sumatorios por separado y luego se han restado. El resto del desarrollo es similar al del filtro FIR.

```
voidiir (Int16 *in, int M, constInt16 *b,constInt16 *a, intntap, Int16 *out,
Int16 *x, Int16 *y)
{
	Int16 j;				// Loop index
	Int32 yn1, yn2;

	for(j=0; j < M; j++)
	{
	yn1=yn2=0;
	x[0] = in[j];		// Insert new data x(n)

	// store convolution result to output buffer
	out[j]= iir_convolution(x, b, ntap) -iir_convolution(y, a, ntap-1);

	shift(x, ntap);			// Shift old data x(n-i)
	shift(y, ntap-1);		// Shift old data y(n-i)

	y[0] = out[j];		// Insert new data y(n)
	}
	return;
}
```

Los parámetros de la función son:

- *in: puntero al buffer de la señal de entrada
- M: es el número de muestras que van a filtrar, en este caso 128.
- *b: puntero al array de coeficientes del numerador
- *a: puntero al array de coeficientes del denominador
- ntap: es el número de coeficientes del numerador
- *out: puntero al array donde se almacenan los datos procesados que posteriormente serán escritos a la salida.
- *x: puntero al buffer donde se almacenan parte de las muestras de la señal de entrada, y que conforme se vayan filtrando se irán desplazando para hacer que todas las muestras pasen por el filtro.
- *y: igual que *x pero para las muestras de salida.

Las funciones iir_convolution y shift son iguales que las que aparecen en la codificación del filtro FIR.

De manera que el programa principal quedaría:

```
voidmain ( )
{
	struct stereo dataIn;
	struct stereo dataOut;

	//Reserve memory
	Int16 *signal_buffer_IN=(Int16 *)calloc(NumLength, sizeof(Int16));
	Int16 *signal_buffer_OUT=(Int16 *)calloc(DenLength, sizeof(Int16));

	BSL_AIC3204_init( );// Initialize CODEC
	BSL_AIC3204_start( );// Start CODEC

	while(1) {// Enter the Main Loop

	BSL_AIC3204_read(&dataIn); // Read 16-bit stereo data

	//block IIR to right channel
	iir(dataIn.R, BUFFER_SIZE, NUM_COEF, DEN_COEF, NumLength, dataOut.R,
		signal_buffer_IN, signal_buffer_OUT);

	BSL_AIC3204_write(&dataOut);// Write 16-bit stereo data
	}
```

Una vez que el filtro está programado ya solo sería necesario incluir el fichero de los coeficientes y ejecutar para comprobar con Scope los resultados obtenidos.

El alumno debe ser capaz de diseñar un filtro de tipo IIR, cargarlo en el DSP y realizar una valoración del resultado obtenido al visualizarlo con Scope.

En esta práctica se le va a dar especial importancia a los problemas de implementación que pueden ir apareciendo, por eso se han planteado al alumno diversas cuestiones relacionadas con esto.

El alumno deberá comparar filtros con las mismas especificaciones pero calculados con diferentes algoritmos (Chebyshev, Butterworth...)

Deberá ser capaz de valorar en qué casos, con la implementación propuesta, es más conveniente usar un filtro de tipo FIR o de tipo IIR, para esto se le pedirá al alumno que haga varias comparaciones de filtros con las mismas características uno de tipo FIR y otro de tipo IIR.

3.5 Práctica 5: TransformadaDiscreta de Fourier

La última práctica la hemos dedicado a la transformada discreta de Fourier. La transformada discreta de Fourier (DFT) tiene un papel muy importante en el diseño, análisis y realización de sistemas y algoritmos de tratamiento de señales en tiempo discreto. Las propiedades básicas de la transformada de Fourier y de la DFT, hacen que analizar y diseñar sistemas en el dominio transformado sea conveniente y práctico. Este hecho, junto con la existencia de algoritmos eficientes para el cálculo explícito de la DFT, la convierte en un componente importante de muchas aplicaciones prácticas de los sistemas en tiempo discreto.

Los objetivos que el alumno deberá alcanzar con la realización de esta práctica son:

- Comprender el funcionamiento de la transformada discreta de Fourier en sistemas de tiempo discreto.
- Deberá ser capaz de elegir un valor de ventana para el cálculo de la DFT acorde a la frecuencia de muestreo del códec y a la frecuencia de la señal de entrada.

Para calcular la DFT de una señal discreta se parte de la siguiente fórmula:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\left(\frac{2\pi}{N}\right)kn}, \qquad k = 0,1,\dots,N-1$$

Como se observa en la ecuación para su resolución es necesario operar con números complejos. Para poder implementar esta ecuación se ha descompuesto en parte real y parte imaginaria, de manera que una vez hecho, se pueda calcular su modulo de forma sencilla para poder representarlo.

La ecuación descompuesta en parte real y parte imaginaria quedaría de la siguiente manera:

$$X(k) = \sum_{n=0}^{N-1} x(n) cos\left(\frac{2\pi kn}{N}\right) - j \sum_{n=0}^{N-1} x(n) sen\left(\frac{2\pi kn}{N}\right)$$

A partir de esta ecuación se ha implementado el programa capaz de pasar la señal de entrada de nuestro DSP del dominio del tiempo al dominio de la frecuencia.

A continuación se muestra como ha sido programada la DFT:

```
voiddft (Int16 *in, Int16 N, Int16 *out)
{
	float real, img, angle;
	Int16 k, n;

		for (k=0; k<N; k++) {

			real = 0.0;
			img = 0.0;

			for(n=0; n<N; n++)
			{
				angle = (2.0*PI*k*n)/(N);

				real+= (float)in[n]*cos(angle);
				img+= (float)in[n]*sin(angle);
			}
			out[k]=(Int16)sqrt((real*real)+(img*img));
		}
}
```

Donde:

- *in: es el puntero al buffer de entrada.
- N: Es el numero de ventanas con el que se va a calcular la DFT, coincide con BUFFER_SIZE.
- *out: es el puntero al buffer de salida.

Como se ha visto con la fórmula anterior se va a calcular la parte real y la parte imaginaria por separado y luego se calculará su modulo para poder representarlo.

Para poder realizar llamar a las funciones sin() y cos() es necesario incluir la librería "math.h".

El programa principal quedaría de la siguiente manera:

```
voidmain ( )
{
	struct stereo dataIn;
	struct stereo dataOut;

		BSL_AIC3204_init( );// Initialize CODEC
		BSL_AIC3204_start( );

		while(1) {    // Enter the Main Loop

	BSL_AIC3204_read(&dataIn);// Read 16-bit stereo data

		dft(dataIn.R, BUFFER_SIZE, dataOut.R);//DFT of dataIn.R

		BSL_AIC3204_write(&dataOut);// Write 16-bit stereo data
		}
}
```

Como se puede observar la programación básica de la DFT es bastante sencilla, el problema viene por la carga computacional que suponen todas estas operaciones.

El cómputo de cada valor X_k de la DFTevaluando directamente la ecuación analítica requiere N multiplicaciones complejas y $(N-1)$ sumas complejas. Para obtener los N valores se necesitan en total N^2 multiplicaciones complejas y $N(N-1)$ sumas complejas. Como casi ningún procesador efectúa operaciones con números complejos, conviene realizar el cálculo con operaciones de números reales, lo que implica que cada multiplicación compleja requiere cuatro multiplicaciones reales y dos sumas reales, y cada suma compleja requiere dos sumas reales.

Por tanto, para cada valor de k, el cálculo directo de X_k necesita $4N$ multiplicaciones reales y $(4N-2)$ sumas reales. Como X_k se debe calcular para N valores diferentes de k, el cálculo directo de la DFT de una sucesión $x[n]$ requiere $4N^2$ multiplicaciones reales y $N(4N-2)$ sumas reales.

Por este motivo, es de vital importancia ajustar bien tanto los valores de la ventana como los valores de la frecuencia de muestreo, ya que dependerá de ellos en gran medida la cantidad de operaciones que se tengan que realizar.

Una vez que el alumno haya superado satisfactoriamente esta práctica deberá ser capaz de comprender el funcionamiento de la DFT y cómo afecta a su

cálculo la F_s y el tamaño de la ventana.

Para alcanzar estos objetivos se ha planteado al alumno que realice una serie de modificaciones en la configuración de los parámetros del programa y del DSP, de manera que pueda analizar las distintas situaciones posibles.

Además, para conocer cómo afectan estos parámetros a la carga computacional del sistema se ha pedido que haga uso de la herramienta de medición de ciclos de reloj.

4 Presupuesto

Para llevar a cabo la ejecución de este proyecto se ha elaborado el siguiente presupuesto que tiene una validez de 15 días desde el momento en el que se entrega.

Concepto	Horas	Precio hora	Total
Diseño de la solución	10	70 €	700 €
Desarrollo	60	40 €	2400 €
Test	18	25 €	450 €
Redacción del manual y la memoria	15	18 €	270 €
Materiales		80 €	80 €
TOTAL			3900 €

*Todos los precios incluyen IVA.

5 Conclusiones

Como se veía en la introducción, este proyecto tiene un objetivo claro, que ha estado presente durante todo el desarrollo a lo largo de sus diferentes fases y que no es otro que implementar un entorno de trabajo basado en DSP con el que los alumnos puedan reforzar y poner en práctica los conocimientos adquiridos en la parte teórica de la asignatura.

Aunque el objetivo final del proyecto se ha alcanzado de una manera bastante satisfactoria, a lo largo de todo el desarrollo han sido varios los puntos en los que se han encontrado dificultades. Por eso, a continuación iremos repasando el desarrollo realizado para comentar aquellos aspectos que nos han parecido más relevantes.

Siguiendo el orden en el que se han ido cubriendo las fases del trabajo, lo primero que se hizo fue elegir el hardware y hacerlo funcionar con el PC, este fue uno de los primeros puntos conflictivos, ya que la versión de Code Composer (v.4) incluida en el kit no funcionaba de la forma esperada y fue necesario hacer uso de una versión superior (v.5), con la que se subsanaron los problemas detectados.

A lo largo de todo el desarrollo se ha podido comprobar que los sistemas reales no siempre siguen un comportamiento como el esperado, ya que aparecen muchos factores que pueden afectar a su funcionamiento. El entorno de trabajo ha sido planteado de manera que uno de los elementos que determinan los resultados es la tarjeta de sonido del PC, ya que tanto la señal generada como la señal adquirida deberán atravesarla añadiendo al sistema un punto de incertidumbre, por lo que en determinados momentos pueden aparecer comportamientos injustificados. Es por eso que a la hora de evaluar los resultados de los diferentes desarrollos, ha sido necesario hacer un uso constante de la herramienta de depuración de Code Composer y realizar repetidas pruebas para poder sacar las conclusiones adecuadas.

Además, se ha podido experimentar como en ocasiones aquello que se ha diseñado y calculado, en este caso con Matlab, a la hora de realizar la implementación no siempre los resultados coinciden con las especificaciones

del diseño. Es, por ejemplo, el caso de la cuantificación de los coeficientes de los filtros. Cuando se diseña un filtro con una especificaciones exigentes, a la hora de cuantificar los coeficientes el sistema puede volverse inestable, ya que no siempre es posible pasar de una aritmética en punto flotante a una aritmética en punto fijo. Además en el caso del sistema diseñado para la implementación del filtro II se ha partido de la forma directa y Matlab lo calcula a partir de secciones de segundo orden, por lo que es necesario operar sobre los coeficientes varias veces hasta que tienen un formato acorde a lo diseñado.

A pesar de todos estos detalles, como se ha dicho en líneas anteriores, con este trabajo se ha conseguido el principal objetivo del proyecto con el que se espera que una vez el alumno haya cursado el laboratorio de la asignatura, haya adquirido una visión más real de lo que supone diseñar e implementar un sistema de procesado digital de señales en tiempo real.

Por último a continuación se van a sugerir una serie de mejoras para posibles trabajos futuros.

En primer lugar, quizá se podría tratar de incluir algún otro contenido teórico además de los ya expuestos.

A partir del trabajo realizado se podrían desarrollar algunas aplicaciones como podría ser un sistema de detección multitono basándose en el algoritmo de Goertzel. Otra mejora que se podría realizar es programar el filtro IIR con secciones de segundo orden para tratar de conseguir mejores resultados de los que se obtienen con la forma directa.

En todas las practicas hemos trabajado con el generador de señales de Scope, se podría plantear trabajar con señales de audio reales para ver cómo se pueden aplicar los conocimientos adquiridos a señales que podemos encontrar en el mundo real.

6 Bibliografía

Esta es la bibliografía básica que se ha empleado durante la elaboración del proyecto:

- Hayes, Monson H., Statistical Digital Signal Processing and Modeling, John Wiley & Sons, 1996.
- Sen M Kuo, Bob H Lee, Real-Time Digital Signal Processing, John Wiley & Sons, 2001.
- John G. Proakis, Dimitris Manolakis: Digital Signal Processing - Principles, Algorithms and Applications. Prentice-Hall, Inc. 1996
- Datasheet y documentación de Texas Instruments así como manuales y ejemplos de uso de CodeComposser.
- 4.- Ayuda de la herramienta Matlab.

7 Anexo

En las páginas siguientes se adjuntan los enunciados de las prácticas que el alumno deberá elaborar como complemento a la asignatura.

PRACTICA 1. EL ENTORNO DE DESARROLLO

OBJETIVOS

- Familiarizarse con la tarjeta de desarrollo TMS320C5515, Scope y el CodeComposer Studio (CCS).
- Conocer las recomendaciones básicas y el procedimiento inicial para utilizar la tarjeta.
- Compilar y depurar un proyecto en CCS y cargarlo en la tarjeta.
- Realizar los ajustes en la configuración de audio de Windows para aprovechar el margen dinámico del códec.

PREPARATIVOS

Conecte la tarjeta con el PC mediante la conexión USB (XDS100 Emu). Esto permite la comunicación de CCS con el DSP para la carga y depuración de código así como otras funciones avanzadas.

Para la comunicación de el códec del PC don el de la tarjeta conecte mediante dos TLR de la entrada “Strereo In” de la tarjeta con la salida de audio del PC y la salida “StereoOut” de la tarjeta con la entrada de línea/micrófono del PC.

DESARROLLO

Un IDE (IntegratedDevelopmentEnvironment), es el conjunto de herramientas hardware y/o software necesarias para implementar una aplicación.

El IDE necesario en este laboratorio consta de:

- Hardware: TMS320C5515 eZDSP USB Stick
- Software: CCS V.5 (CodeComposer Studio) y Scope (Generador de señales y osciloscopio).

TMS320C5515 eZDSP USB Stick

El C5515 es un procesador de señales digitales (DSP) que forma parte de la familia C55 de DSPs en punto fijo de Texas Instruments (TI) diseñados especialmente para el desarrollo de aplicaciones de bajo consumo.

Las siglas DSP (Digital SignalProcessoro procesador digital de señales) hacen

referencia a microprocesadores que poseen un juego de instrucciones, un hardware y un software optimizados para aplicaciones que requieren operaciones numéricas a muy alta velocidad.

Es un procesador diseñado para manipular señales digitales en tiempo real mediante secuencias de instrucciones iterativas a muy alta velocidad. La diferencia esencial entre un DSP y un microprocesador es que el DSP tiene características diseñadas para soportar tareas de altas prestaciones, repetitivas y numéricamente intensas. Los DSPs se utilizan habitualmente para medir, filtrar y/o comprimir señales analógicas (ej. filtrar y convertir audio en diferentes formatos). Por tanto, el primer paso para realizar el tratamiento de señales consiste en convertir las señales analógicas en un flujo de muestras digitales a través de un ADC (Analog/Digital Converter).

Cuestión 1: Complete esta tabla con las características más relevantes del DSP de la tarjeta (consulte en la web de TI):

	TMS320C5515
DSP	
Tipo de instrucciones	
DSP MHz (Max.)	
General Purpose Memory	
# USBs	
# MMC/SD	
# UART (SCI)	
# ADC	
# I2C	
# SPI	
# DMA (Ch)	

La tarjeta TMS320C5515 eZDSP USB Stick de Spectrum Digital, confiere al DSP C5515 varios módulos periféricos como son el conector de expansión de 40 pines, un puerto de expansión para un módulo bluetooth, 1 puerto USB 2.0, ranura para lector de tarjetas microSD, dos pulsadores, un conjunto de LEDs y un códec de audio programable de 32 Bits modelo TLV320AIC3204 de TI que

tiene en sus puertos de entrada/salida un conector estéreo de audio tipo mini Jack o TLR. La tarjeta también dispone de un bus serie que permite la conexión con un equipo host para transmitir a través del emulador embebido del XDS100 JTAG el programa de carga y depuración del DSP.

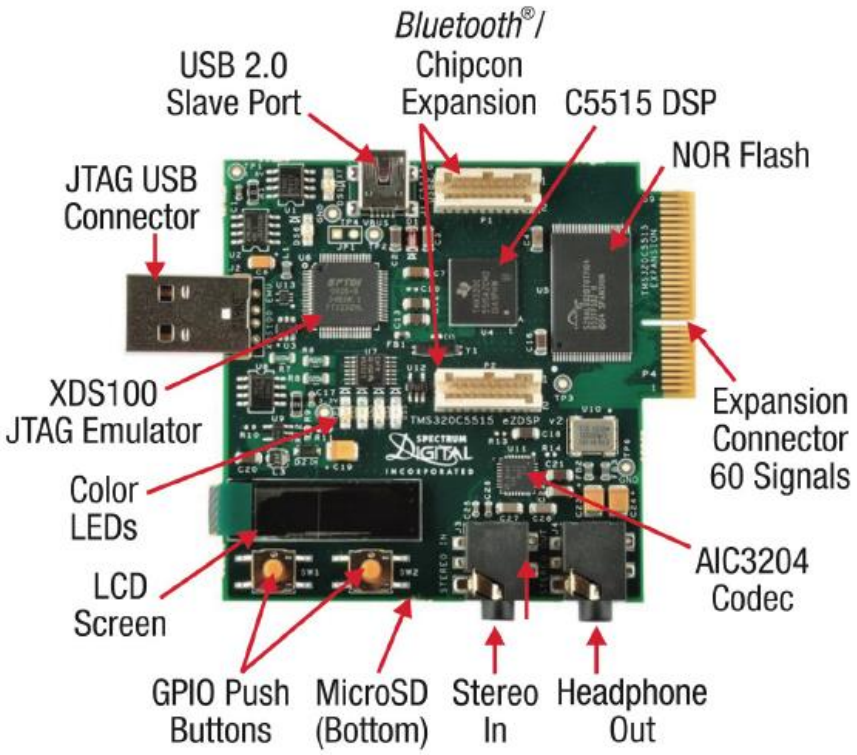

Ilustración 1. Detalle de la tarjeta de desarrollo TMS320C5515

Cuestión 2: Complete esta tabla con las características más relevantes del códec de la tarjeta (consulte en la web de TI):

	TLV320AIC3204
# DACs	
# ADCs	
# Inputs / # Outputs	
Resolución (Bits)	
Frecuencia de muestreo (Max) (MHz)	
Protocolos para la interfaz de control	

CODE COMPOSER STUDIO

Para poder implementar una aplicación para el DSP son necesarias cuatro herramientas básicas: editor de código fuente, compilador, enlazador y depurador. Estas herramientas, más un conjunto muy superior de otras utilidades, son las que conforman CodeComposer Studio (CCS).

La creación de un nuevo proyecto requiere la realización de diversos pasos. Hay que tener en cuenta que cada proyecto debe de tener un nombre único, lo que permite abrir varios proyectos simultáneamente. La información de cada proyecto se guarda en un entorno común llamado workspace. Durante las prácticas se le facilitara un workspace y un conjunto de proyectos para el desarrollo de la misma por el alumno.

Cuando se abre CCS por primera vez, el programa debería solicitar al usuario el área de trabajo (workspace) a utilizar, que es la ruta física donde van a colgar los proyectos. En la ilustración 2 se puede ver la ventana emergente para dicha operación.

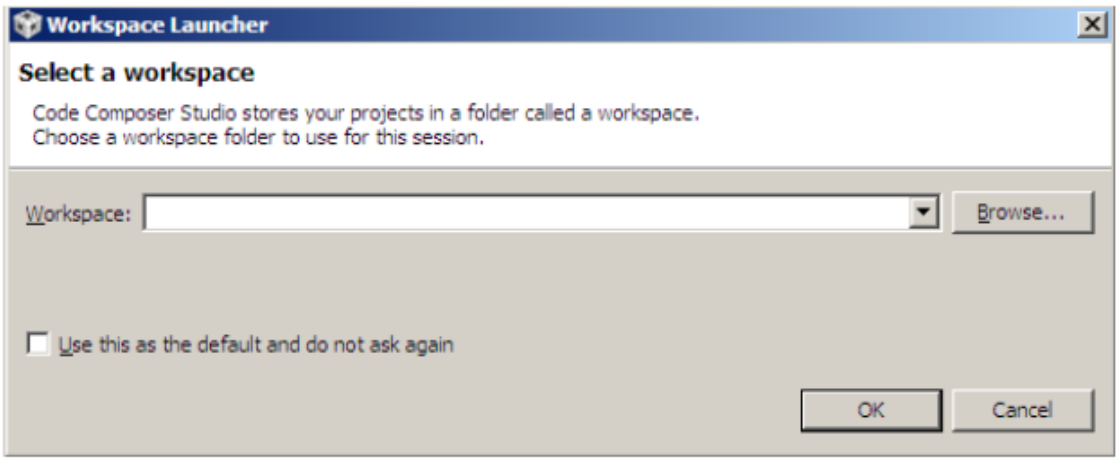

Ilustración 2. Selección del área de trabajo

Seleccione el workspacede nombre “Practica 01” facilitado para esta práctica (Se recomienda no marcar la casilla “Use this as default and do no taskagain” para que dicha ventana aparezca en cada sesión de laboratorio aunque se puede llegar a este lugar navegando por las opciones del CCS.) Se debería abrir una ventana de CCS con el aspecto mostrado en la ilustración 3.

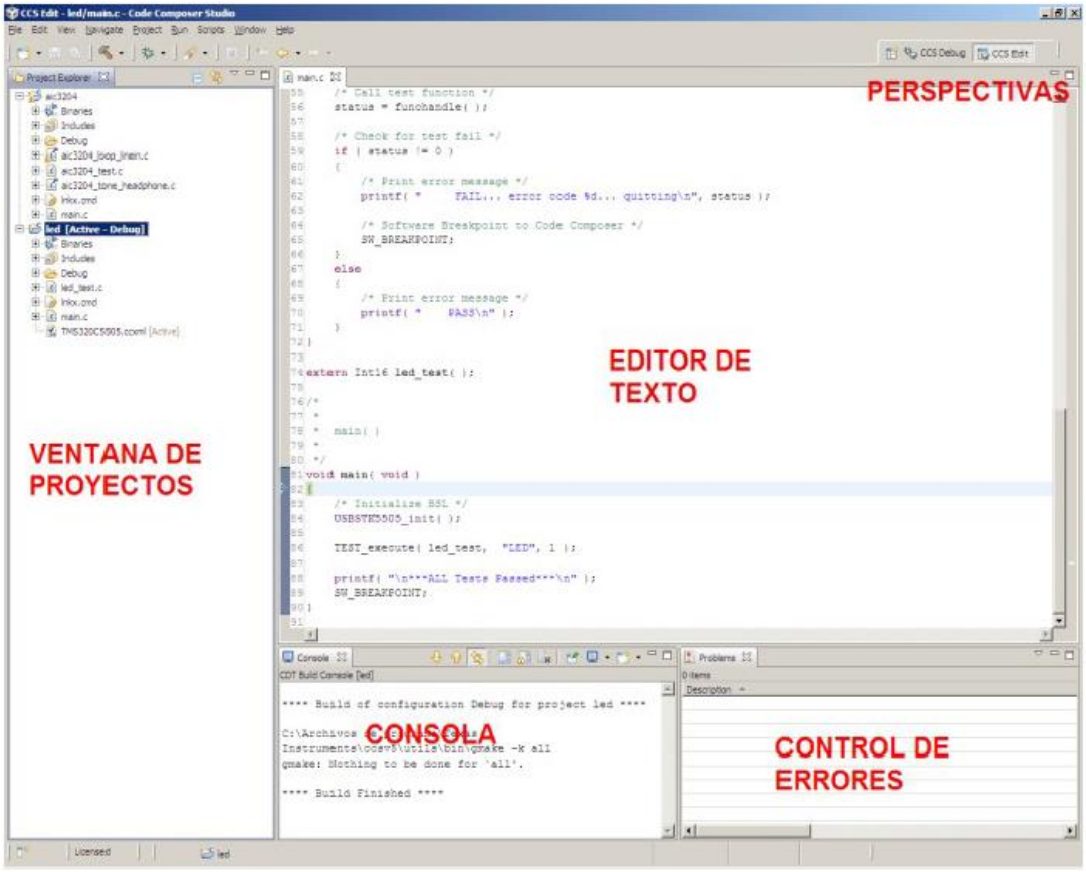

Ilustración 3. Perspectiva de edición

En CCS, un proyecto está compuesto por diferentes tipos de archivos. Estos son necesarios para poder generar un único programa ejecutable. Existen siete tipos de archivos principales:

- Código fuente en C (Extensión *.c)
- Código fuente en ensamblador (Extensión *.asm)
- Documentos de cabecera en C (Extensión *.h)
- Bibliotecas de enlazado externas (Extensión *.lib)
- Mapa de memoria del DSP (Extensión .cmd)
- Fichero de configuración de la tarjeta con CCS (Extensión *.ccxml)
- Script de arranque de la tarjeta en lenguaje GEL. (Extensión *.gel)

Explore el proyecto “talkthrough” para entender la jerarquía establecida por el CCS. Localice la función “main()” en el fichero “main.c” e intente seguir el flujo del programa aproximado. Este hábito es recomendable cuando se comienza cualquier proyecto a partir de fuentes existentes para hacerse una composición de lugar.

Breve descripción de la estructura

*El programa que se va a usar en TODAS las prácticas esta implementado siguiendo un patrón de diseño "*chain of command*". Este patrón desacopla el emisor de una petición del receptor.*

*Siguiendo este patrón, el programa está estructurado en 3 partes bien diferenciadas: CSL (*Chip Support Library*), BSL (BoardSuppor Library) y el núcleo del programa. La idea de estas tres secciones es:*

CSL: Enmascarar todos los módulos y funcionalidades del DSP en llamadas del tipo (init, open, close, config, read y/o write). Se entrega como biblioteca precompilada con extensión (.lib*) para su enlace en el proyecto con el cual se está trabajando. El proyecto mediante el cual se ha realizado dicha biblioteca se encuentra en cada* workspace*que se entrega al alumno al comienzo de cada práctica y con nombre "CSL C5515 v2_5". No se debe manipular este proyecto en ningún aspecto. Sólo se entrega como consulta y para que el alumno curioso y/o interesado pueda indagar en el funcionamiento interno del sistema.*

*BSL: Enmascarar todos los periféricos de la tarjeta de desarrollo en un conjunto de llamadas del tipo (init, open, close, config, read y/o write). Se entrega como un conjunto de ficheros de código fuente en C y ficheros de cabecera, con prefijo (*BSL*), para su uso en el núcleo del programa. Entre estos ficheros, se encuentra el encargado de enmascarar las funcionalidades del códec, el cual se usará en la próxima practica para variar la frecuencia de muestreo.*

NÚCLEO: Embebido en la función main() *aquí se realizan las llamadaspertinentes para configurar los diferentes periféricos haciendo uso de llamadasa la capa BSL y se ejecuta el procesado principal de la aplicación(normalmente encerrado en un bucle infinito).*

Siguiendo con el funcionamiento del CCS, en el marco superior existen dos herramientas fundamentales para la creación y depuración de proyectos (ver ilustración 4). El primer control (Build) inicia la compilación del proyecto. El segundo control (Debug) no sólo compila, además carga en la tarjeta el binario generado como resultado de una compilación satisfactoria.

Ilustración 4. Iconos de compilación y depuración respectivamente.

Seleccione el proyecto "talkthrough" en la pestaña de proyectos, el nombre

resaltará en negrita. Esto indica a CCS que éste proyecto es el proyecto activo sobre el que se está trabajando. Ejecute el control para iniciar la depuración. SiCCS finaliza de forma satisfactoria, el resultado será la ventana que se muestra en la ilustración 5.

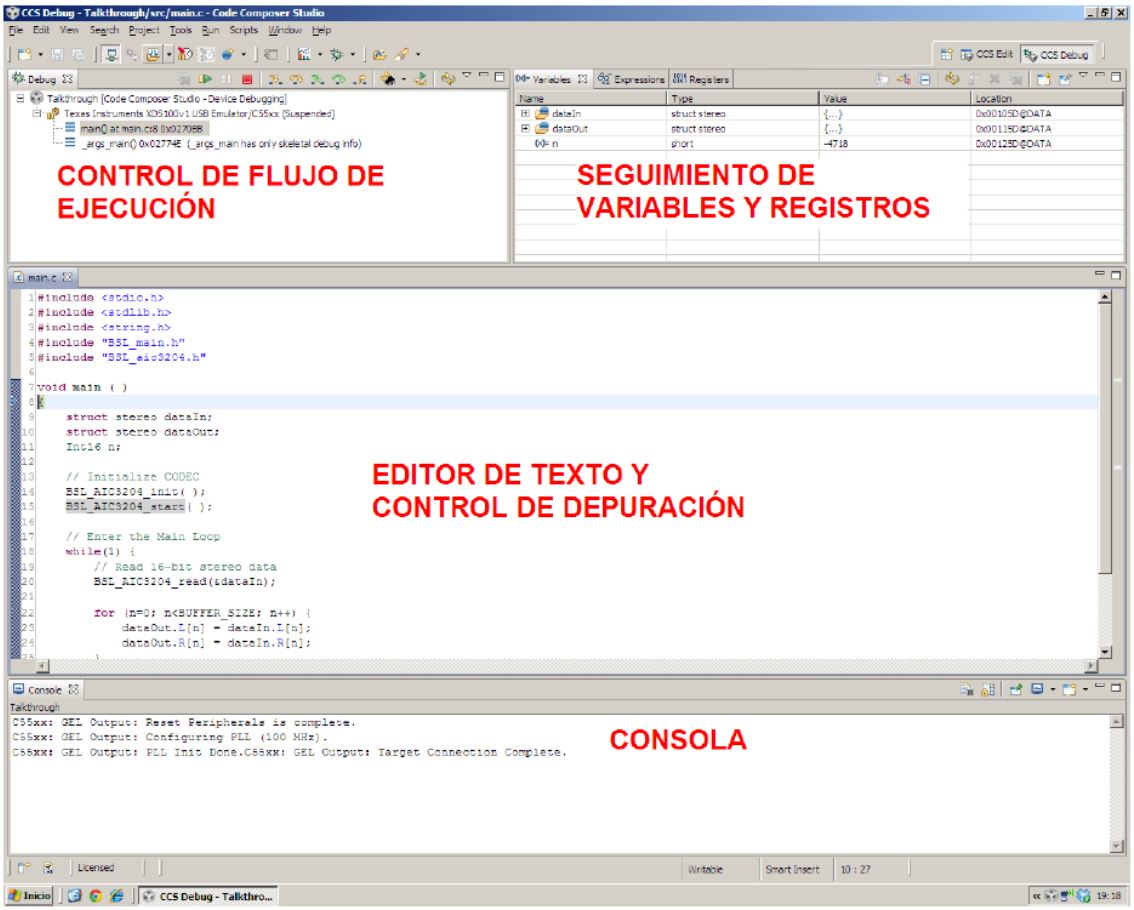

Ilustración 5. Perspectiva de depuración.

CCS está basado en Eclipse y por ello hereda el modo de trabajo que se conoce como “Perspectivas”. Una perspectiva hace referencia al conjunto de ventanas y controles, denominados módulos, que son visibles en pantalla. Las dos perspectivas más comunes son Edición (ilustración 3) y Depuración (ilustración 5).

El programa debería estar detenido al comienzo de la función main(). Seprocede a ejecutar el programa pulsando en el botón resume (Botón de “play”, ilustración 6).

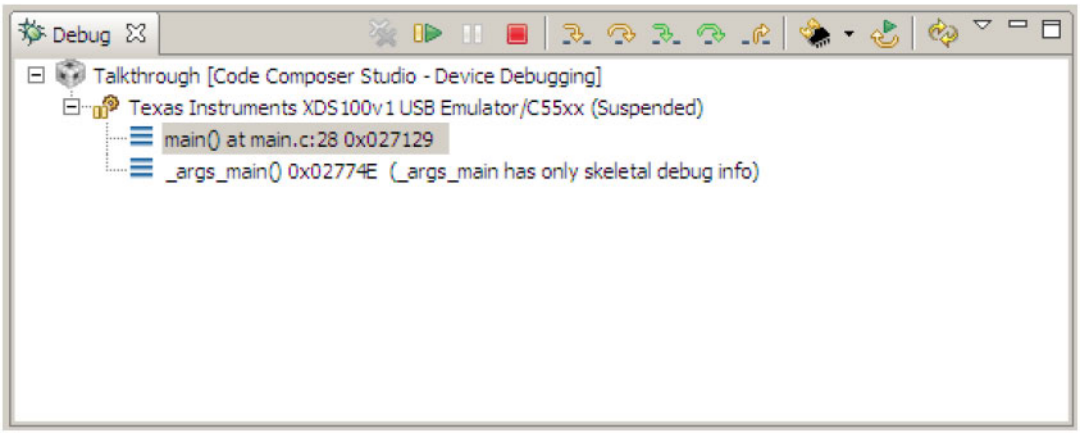

Ilustración 6. Controles de flujo de ejecución

Pulse el botón de pausa. El programa se detendrá en el lugar donde apunte el contador de programa. Este lugar no tiene porque tener su correspondencia con un fichero de código fuente en el proyecto y por tanto puede que ese lugar no sea visible por el editor.

Ahora pruebe a colocar algún punto de ruptura (doble click en la barra de numeración de línea del editor) y a navegar por el código paso a paso haciendo uso de los controles en la barra de herramientas o con sus atajos de teclado. Si tiene alguna duda consulte con el profesor del laboratorio.

Aventúrese a añadir o eliminar alguna vista y modificar el posicionamiento de los módulos del CCS a su gusto. Cuando se cierre el workspace todos los cambios realizados se guardarán.

SCOPE

Otra de las herramientas que conforman el IDE es el programa Scope de uso gratuito y sin restricciones si se hace un uso particular y educativo del mismo. Ésta aplicación está formada por 6 módulos: un osciloscopio, un analizador XY, un analizador de espectros, un generador de señales, un módulo configuración y extras. La herramienta resulta bastante intuitiva. Si encuentra alguna dificultad en su uso puede dirigirse a la documentación que se encuentra en la carpeta del programa.

Ejecute el programa Scope y seleccione el módulo del generador de señales (SignalGenerator). Debería visualizar el módulo que se observa en la ilustración 7. Configure ambos canales para generar una señal de 1 Vpp de

100Hz y 1kHz respectivamente desfasadas 90 grados y actívelos.

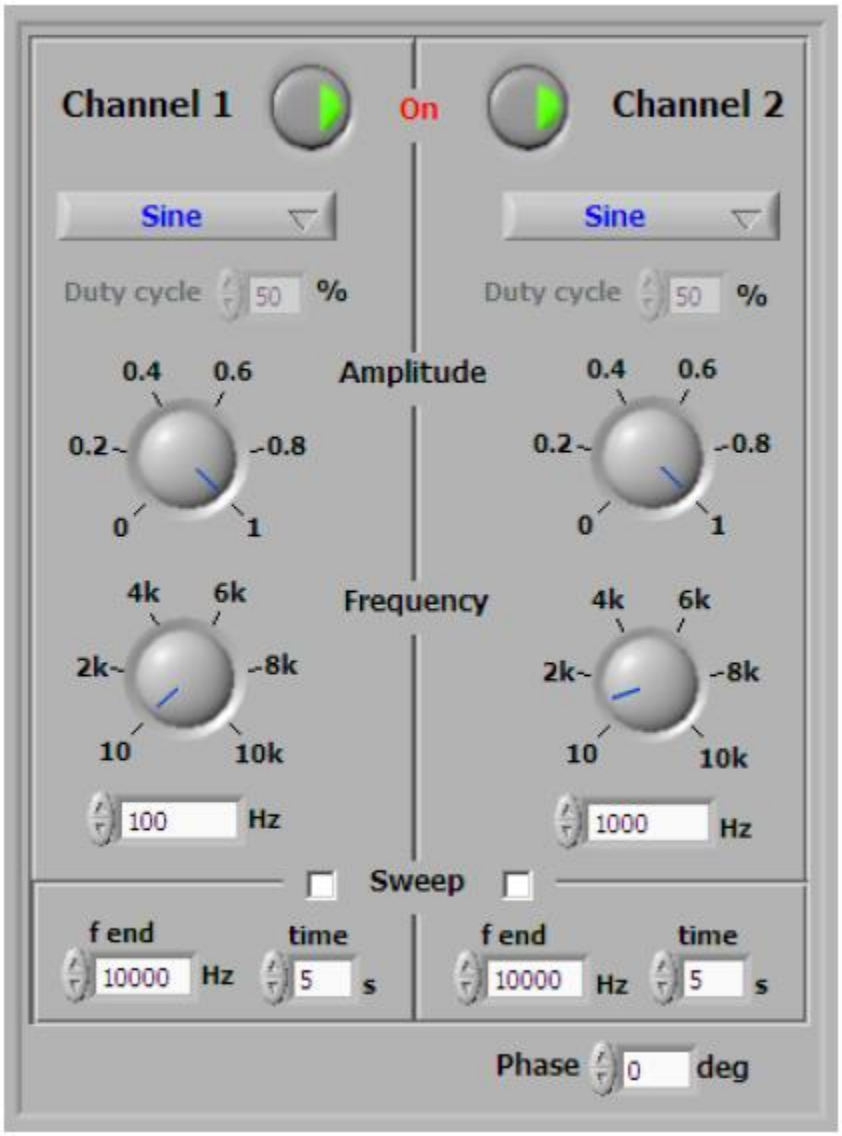

Ilustración 7. Módulo generador de señales de Scope.

Retome la sesión en CCS para realizar una depuración avanzada, para lo cual se va a hacer uso de una herramienta muy útil: las gráficas.

Se pide detener el código haciendo uso de un punto de ruptura justo antes de la escritura en el códec (en la llamada a la función “AIC3204_write” en la linea28 del fichero “main.c”).

El objetivo de parar en esta línea de código, es que en este punto se tiene en la memoria la estructura “dataIn” con el bloque de muestras capturadas por el códec para el canal izquierdo y derecho en la iteración actual. Dicha estructura

está segmentada en dos vectores (L y R) de 128 muestras de 16 Bits (ver Ilustración 8). La definición de dicha estructura y el tamaño de los vectores se puede encontrar en el fichero “BSL_main.h”.

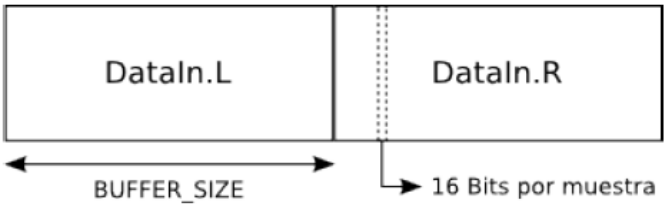

Ilustración 8. Módulo generador de señales de Scope.

Haga uso de la herramienta “Tools/Graph/Dual Time”, se abrirá una ventana como la de la ilustración 9. Configure los parámetros como se muestran en dicha ilustración.

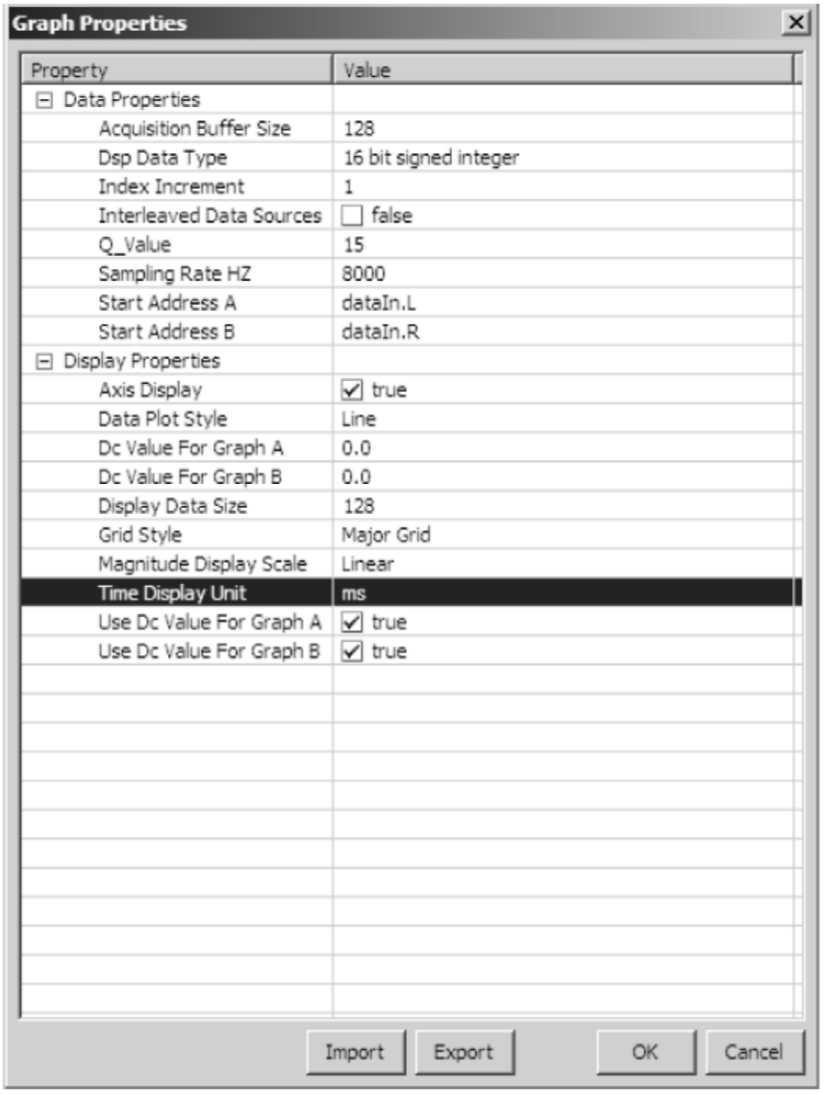

Ilustración 9. Ventana de opciones para graficas en modo single-time.

La configuración de las propiedades de los datos, “Data Properties”, se ha realizado atendiendo al siguiente criterio:

- Acquisition buffer size: 128 muestras capturadas por cada canal.
- DSP Data type: 16 Bits pormuestra.
- IndexIncrement: Los saltos entre muestras son de una en una. Las muestras de cada vector (canal L y R) se capturan y almacenan deforma consecutiva en memoria.
- Interleaved Data Sources: Es falso por lo comentado en el punto anterior.
- Q_value: Formato de las muestras (1Q15). De los 16 Bits el bit más significativo se utiliza para la parte entera y los bits restantes se utilizan como parte decimal del número. El objetivo de este parámetro es otorgarle al eje de las ordenadas (Eje Y) una correspondencia con el valor real en tensión de cada muestra capturada por el códec. Se explicará de forma más detallada en la práctica 2.
- SamplingRate HZ: El códec está configurado para capturar muestras a 8kHz.
- StartAddress A/B: Las direcciones de comienzo de los vectores a representar en la gráfica A o B.

Para las propiedades de visualización, “DisplayProperties”, la configuración se realiza atendiendo al siguiente criterio:

- Axis Display: Verdadero puesto que queremos mostrar los ejes.
- Data Plot Style: Las muestras están unidas por una línea.
- DC Value for Graph A/B: Sin offset. El valor de la componente continua para ambos canales es de 0V.
- Display Data Size: 128 muestras a mostrar en el gráfico.
- Grid Style: Solo se muestra la rejilla para el eje de abscisas y ordenadas.
- MagnitudeDisplayScale: El eje de ordenadas (Eje Y) está representado en escala lineal.
- Time DisplayUnit: Gracias a que configuramos el parámetro “SamplingRate Hz” en las propiedades de los datos, CCS puede convertir el eje de abscisas en unidades temporales (milisegundos) en lugar de muestras(samples).

- “DC ValueforGraph A/B”: Ambos están activos para poder manipular losvalores “DC ValueforGraph A/B”.

Ejecute el código varias veces parando en el punto de ruptura establecido anteriormente. En cada iteración, CCS actualiza los datos de dos nuevas vistas(DualTimeA-* y DualTimeB-*) en la perspectiva de depuración como la mostrada en la ilustración 10. En caso de no obtener una señal como la mostrada en la ilustración pruebe a realizar los ajustes en las propiedades de audio de Windows y compruebe el conexionado.

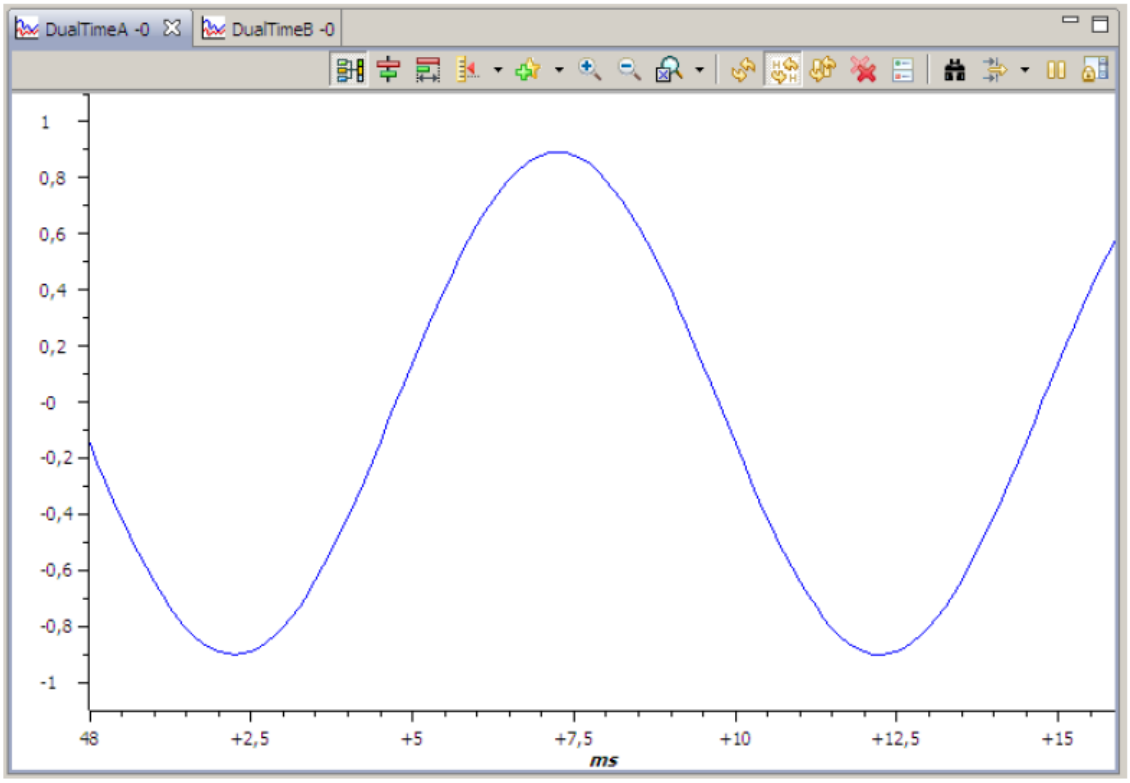

Ilustración 10. Módulo de depuración de vectores en modo gráfico.

Trate de aprovechar al máximo las prestaciones del códec haciendo uso de todo el margen dinámico de la entrada. Para ello es necesario abrir las opciones de audio de Windows y ajustar el volumen de salida de los altavoces, hasta lograr, mediante ejecuciones a distintos volúmenes, que la señal representada en la gráfica aproveche el margen de 1 Vpp del códec sin saturar (Recortes en los máximos y mínimos de la señal sinusoidal).

Con esta tarea además de aprovechar el margen dinámico del códec se ha logrado que el valor del potenciómetro de amplitud de Scope esté ajustado

linealmente con el valor de cada muestra en la gráfica de CCS. Por este motivoes por el cual se fija a nivel máximo el generador de señales de Scope y se ajusta el volumen de Windows, y no a la inversa.

Es recomendable realizar esta tarea en cada práctica en caso de utilizar un ordenador del laboratorio (muchas personas hacen uso del mismo puesto y pueden modificar estos parámetros).

Cuestión 3: Capture un conjunto de imágenes para el canal L y R (Graph A/B) en el que se muestre la señal configurada en el apartado anterior de 1Vpp a100 Hz y 1kHz sin saturar y aprovechando el margen dinámico.

Elimine el punto de ruptura y continúe el programa para que se ejecute libremente, sin interrupción. Recupere la sesión con Scope y seleccione el módulo del osciloscopio (Oscilloscope). Ajuste los parámetros del osciloscopio en 250mV/Div (Offset 0) y con 10ms/Div para ambos canales (Utilice la función sync para acoplar ambos canales).

En esta ocasión vamos a ajustar la entrada de línea/micrófono de Windows. Sila señal está saturada, pruebe a atenuar el volumen de la entrada. Si esto nofuera suficiente para evitar la saturación compruebe en las opciones avanzadas de la tarjeta de audio de Windows (Consulte con el profesor).

En ocasiones existe un parámetro de configuración en Windows para dar ganancia a la entrada de línea/micrófono cuando se usa la entrada en modo micrófono ya que estas señales son de muy baja amplitud y requieren de una ganancia previa. Si está en uso, desactívela.

Una vez obtenidas las señales sin distorsión, como se muestra en la ilustración11, investigue como activar los cursores del módulo del osciloscopio y mida mediante ellos las frecuencias y amplitudes de estas dos señales.

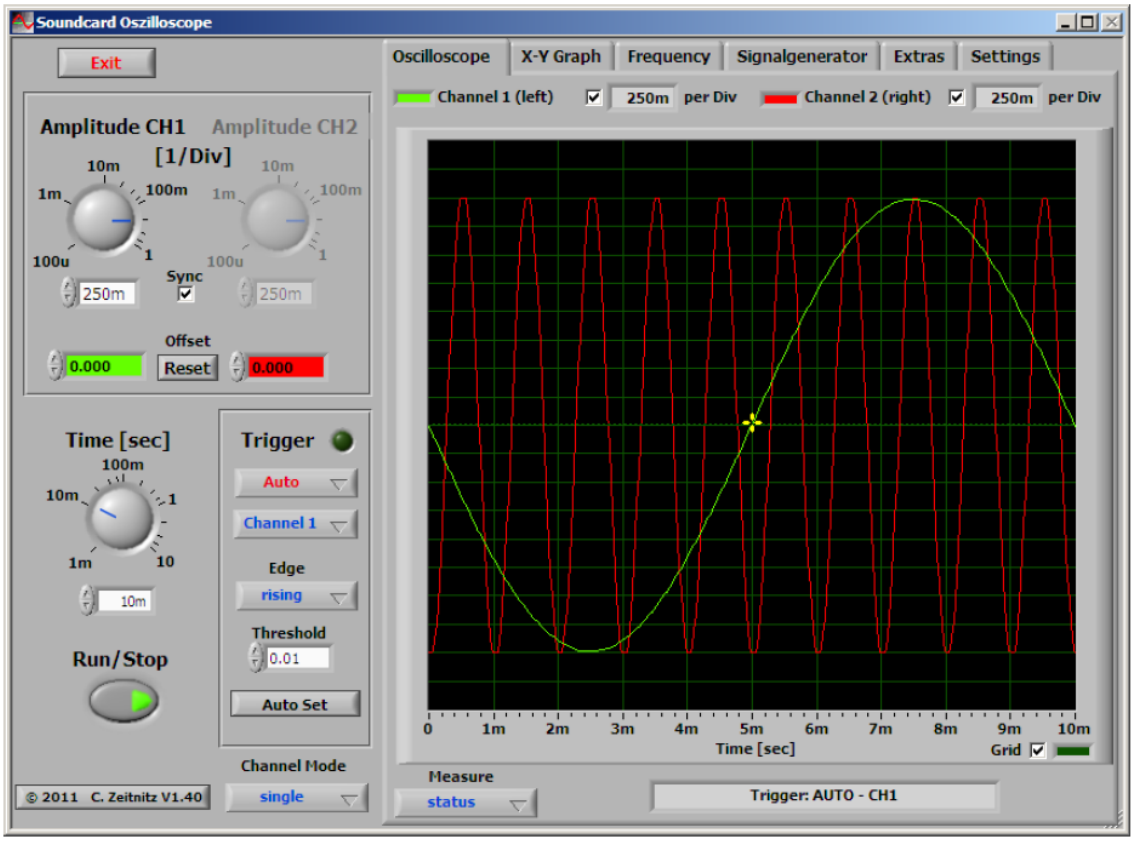

Ilustración 11. Módulo de depuración de vectores en modo gráfico.

Cuestión 4: Capture una imagen para cada señal generada en la que se muestren 10 periodos en pantalla para el canal 2 (canal R) y los cursores delimitando los parámetros fundamentales de la señal (amplitud y periodo).

Cuestión 5: Seleccione el módulo de análisis espectral (Frequency) y configure el módulo para que reciba la información del canal 1 y 2 (canal L y R) y muestreel resultado en dB. Capture una imagen del resultado obtenido en el rango defrecuencias de 0 a 4kHz.

Cuestión 6: Ahora visualice únicamente el canal 2 (Canal R). ¿Cuál es laamplitud máxima que logra obtener con una distorsión armónica aproximadadel 0.1% para este canal?

Cuestión Opcional: Al igual que se ha obtenido el espectro de la señal a la entrada mediante el uso de la herramienta Scope, se puede conseguir estemismo resultado de análisis en frecuencia mediante el uso de gráficas de CCS,como se mostró en apartados anteriores en el dominio del tiempo.

Haga capturas del módulo de la FFT de un bloque de 128, 256 y 1024 muestras para cada uno de los canales. Haga una tabla con los parámetros que ha usado para representar el módulo de la FFT y explique por qué ha elegido estos valores.

PRÁCTICA 2. MUESTREO

OBJETIVOS

- Conocer y manipular los parámetros básicos de configuración del códec.
- Analizar cuáles son los efectos del muestreo sobre la señal.
- Analizar el efecto submuestreo.
- Manejar cada canal (R y L) independientemente.

PREPARATIVOS

Conecte la tarjeta con el PC mediante la conexión USB (XDS100 Emu). Estopermite la comunicación de CCS con el DSP para la carga y depuración decódigo así como otras funciones avanzadas.

Para la comunicación del códec del PC don el de la tarjeta conecte mediantedos TLR de la entrada “Strereo In” de la tarjeta con la salida de audio del PC yla salida “StereoOut” de la tarjeta con la entrada de línea/micrófono del PC.

Cargue el proyecto de la práctica y genere con Scope una señal sinusoidal de1kHz@1V en ambos canales. Compruebe que los ajustes de volumen deentrada y salida del PC están calibrados correctamente. Si no existiera unacorrespondencia entre el volumen asignado en Scope y el volumen de salida serequiere una calibración de la entrada y salida de audio del PC. Si no recuerdaeste procedimiento, siga los pasos indicados en la práctica 1 para realizar dichoajuste y trate de afianzar este procedimiento (o memorizar los ajustes) puestoque será necesario realizar esta tarea antes de comenzar cada práctica.

Para evitar este problema, opcionalmente el alumno podrá usar su ordenadorpersonal para el desarrollo de las práctica y de esta forma asegurar que losparámetros de volumen no queden modificados entre las diferentes prácticasen el transcurso del curso. Solicite al profesor el documento de instalación deCCS si así lo decidiera.

DESARROLLO

Inicie CCS y seleccione el workspace de nombre “Practica 02” facilitado paraesta práctica (Se recuerda que es recomendable NO marcar la casilla para quedicha ventana aparezca en cada sesión de laboratorio aunque se puede llegara este lugar navegando por las opciones del CCS).

Compile y cargue el programa siguiendo el método estudiado en la práctica 1.Ejecute Scope y navegue al generador de señales. Se pide, generar una señalsinusoidal de amplitud igual a 1V y frecuencia igual a 1kHz para el canalizquierdo y la misma señal desfasada 90º para el canal derecho.

Cuestión 1: Capture una imagen haciendo uso del osciloscopio de Scope paracada señal generada en la que se muestren 10 periodos en pantalla y loscursores delimitando los parámetros fundamentales de la señal (amplitud yperiodo).

Cuestión 2: Capture una imagen en modo XY. Explique los resultados.

Cuestión 3: Analice la respuesta en frecuencia de la tarjeta haciendo uso deruido blanco (Configure el analizador de espectros en dB y modo PeakHold).¿A qué frecuencia de muestreo está configurada?

Cuestión 4: Utilice ahora señales cuadradas como entrada. Observe la salidade la tarjeta a diferentes frecuencias (Se recomienda probar con frecuenciasmuy alejadas y próximas a la frecuencia máxima impuesta por el muestreo).¿Qué conclusiones saca tanto en el dominio del tiempo como en el de lafrecuencia? Adjunte las capturas que considere necesarias explicando losresultados.

La frecuencia de muestreo del códec puede configurarse cambiando algunaslíneas de código del fichero “BSL_aic3204.c”. A continuación se muestra unasección de la función “BSL_AIC3204_init()”. Encuentre las líneas de códigosiguientes:

```
// CODEC_CLKIN Configuration (w/ PLL)
```

```
AIC3204_rset( 0, 0x00 );// Select page 0
AIC3204_rset( 4, 0x03 ); // MCLK -> PLL -> CODEC_CLKIN
AIC3204_rset( 6, 0x07 );// J = 7
AIC3204_rset( 7, 0x06 ); // D - HI_BYTE = 1680 (0x0690)
AIC3204_rset( 8, 0x90 ); // D - LO_BYTE = 1680 (0x0690)
AIC3204_rset( 5, 0x91 );// P = 1 | R = 1 | Power up PLL

// ADC_FS & DAC_FS Configuration
AIC3204_rset( 0, 0x00 ); // Select page 0
AIC3204_rset( 13, 0x00 );// DOSR - Hi_Byte = 128 (0x0080)
AIC3204_rset( 14, 0x80 ); // DOSR - Lo_Byte = 128 (0x0080)
AIC3204_rset( 20, 0x80 ); // AOSR = 128 (0x0080) - Filter A
AIC3204_rset( 11, 0x8C ); // Power up and set NDAC value to C
AIC3204_rset( 12, 0x87 ); // Power up and set MDAC value to 7
AIC3204_rset( 18, 0x8C );// Power up and set NADC value to C
AIC3204_rset( 19, 0x87 ); // Power up and set MADC value to 7
```

La función “BSL_AIC3204_rset (dir, val)” se utiliza para configurar los registros del códec. Esta función requiere dos parámetros: la dirección y el valor del registro del códec que se quiere modificar.

Cuestión 5: Experimente con los siguientes valores y complete la tabla:

Configuración	1	2	3	4	5	6
D – High Byte (Reg 7)	0x06	0x06	0x06	0x06	0x06	0x02
D –Low Byte (Reg 8)	0x90	0x90	0x90	0x90	0x90	0x90
NDAC (Reg 11)	0x02	0x04	0x08	0x0A	0x0C	0x03
MDAC (Reg 12)	0x07	0x07	0x07	0x07	0x07	0x05
NADC (Reg 18)	0x07	0x07	0x07	0x07	0x07	0x05
MADC (Reg 19)	0x02	0x04	0x08	0x0A	0x0C	0x03
Fs (KHz)						

Cuestión 6: Ahora ensaye con los siguientes valores:

D – High Byte (Reg 7)	0x06
D –Low Byte (Reg 8)	0x90
NDAC (Reg 11)	0x0C
MDAC (Reg 12)	0x07
NADC (Reg 18)	0x07
MADC (Reg 19)	0x02

Estos valores configuran la frecuencia de muestreo del ADC a 48 KHz pero

ladel DAC a 8KHz. Utilizando esta configuración varíe lentamente la frecuenciade una onda sinusoidal entre 0Hz y 24 KHz anote sus conclusiones.

Cuestión 7: Vuelva a configurar el códec a 48kHz (ADC y DAC). Modifique elprograma para que por uno de los canales de salida muestre la señalsinusoidal que se adquiere, pero por el otro saque una señal cuadrada de lamisma frecuencia que la señal sinusoidal. Anote sus comentarios y adjunte con la práctica el código creado.

Nota: En la función main se dispone de la estructura dataIn que contiene la información delcanal izquierdo y derecho en arrays de tamaño BUFFER_SIZE. Utilice la estructura dataOutpara escribir las muestras a la salida de la tarjeta.

En el fichero de cabecera “BSL_main.h” se encuentra la definición de la estructura estéreo ydel tamaño del buffer.

```
#define BUFFER_SIZE 128
structstereo {

        Int16 L[BUFFER_SIZE];
        Int16 R[BUFFER_SIZE];

};
```

Cuestión 8: Ahora modifique el programa para que por un canal salga la señalsinusoidal adquirida y por el otro una señal diente de sierra de frecuenciafundamental 2 KHz como la mostrada en la ilustración 2. Anote susconclusiones.

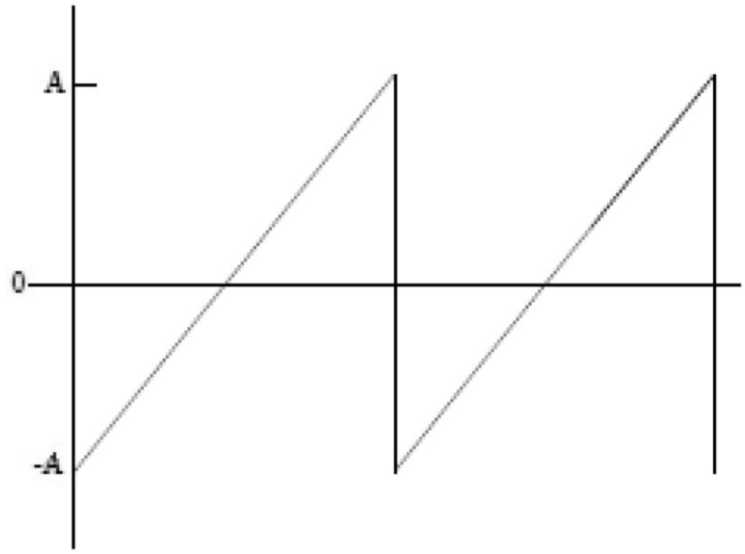

Ilustración 1. Señal de diente de sierra.

PRÁCTICA 3. FILTRADO FIR

OBJETIVOS

- Diseñar y caracterizar un filtro FIR con ayuda de Matlab, incluyendo el cálculo de los coeficientes.
- Exportar desde Matlab los coeficientes a CCS para su implementación.
- Interpretar los resultados obtenidos una vez que haya conseguido implementar el filtro diseñado en el DSP.

DESARROLLO

Para poder implementar un filtro FIR en nuestro DSP, lo primero que debe hacer es calcular los coeficiente del filtro. Para ello inicie Matlab y ejecute la herramienta Fdatool (bastará con ponerlo en el prompt para lanzarla). El funcionamiento de Fdatool es muy sencillo e intuitivo por lo que no debería suponer un problema su uso. En cualquier caso puede consultar la ayuda de Matlab.

Cuestión 1: Diseñe un filtro paso bajo tipo "FIREquiripple" con las siguientes especificaciones:

- Fs: 48 KHz
- Fpass: 4 KHz
- Fstop: 5 KHz
- Apass: 1 dB
- Astop: 80 dB

Una vez haya diseñado el filtro,exporte los coeficientes a un fichero .h, para incluirlo en el proyecto de CCS que se va a utilizar para la práctica.

Para exportar los coeficientes vaya a Targets ->Generate C Header...

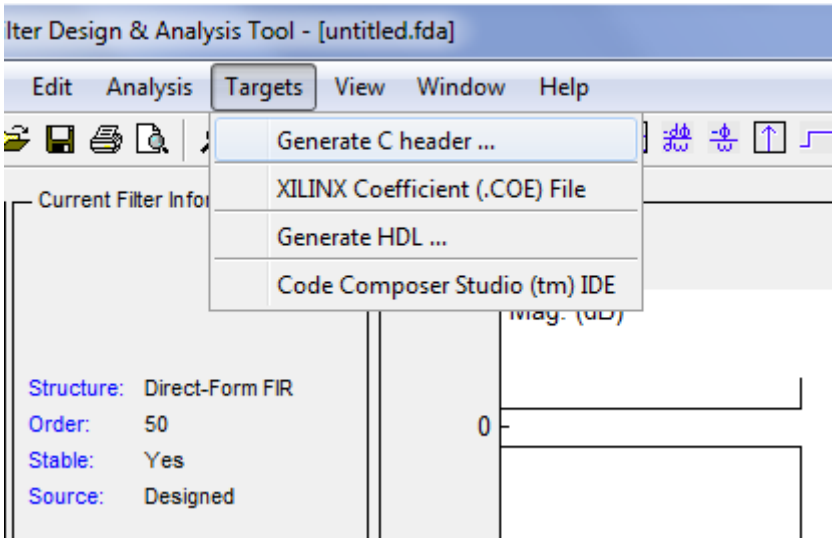

Ilustración 1. Generar fichero de cabecera '.h'.

Se abrirá una ventana como la siguiente y deberá rellenar los campos tal y como se indica en la imagen

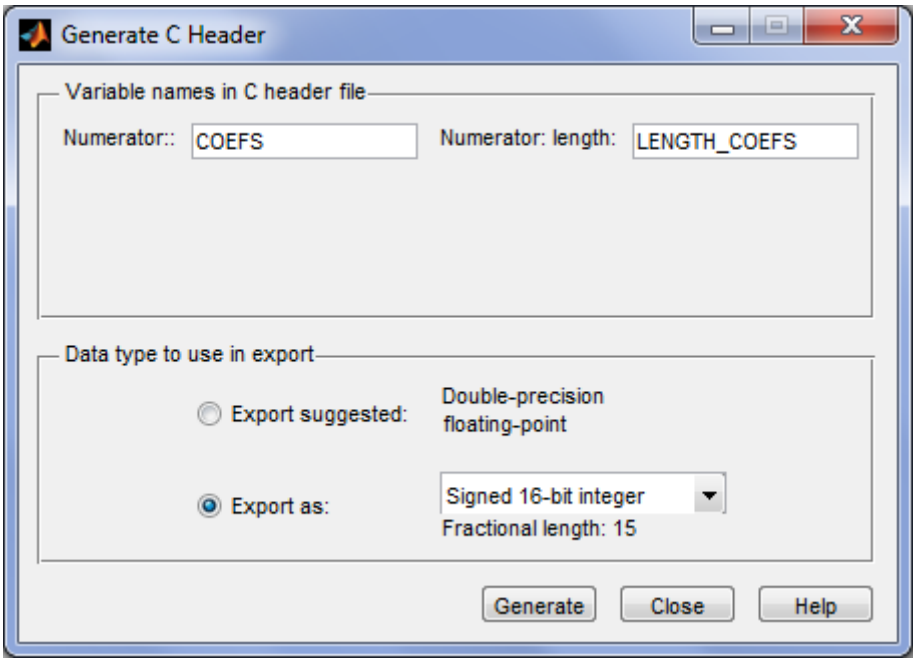

Ilustración 2. Definición de parámetros del fichero '.h'.

Es importante que ponga exactamente los mismos nombres, ya que con ese nombre serán referidos los coeficientes del filtro en el proyecto de CodeComposer.El fichero se deberá guardar dentro del workspace Práctica 3, en la carpeta "include" del proyecto.

Recomendación: Dentro del fichero .h generado, se recomienda escribir como comentario las especificaciones del filtro. De esa manera, podrá saber de qué

filtro se trata en caso de confundir los ficheros.

Inicie CCS y seleccione el workspace de nombre “Practica 03” facilitado para esta práctica. Conecte la salida de audio del PC a la entrada de la tarjeta. Conecte la salida de la tarjeta a un la entrada de línea del PC. Asegúrese de que el fichero ".h" con los coeficientes se encuentra dentro de su proyecto. Una vez verificado, modifique el programa principal (main.c) e incluya la cabecera del fichero ".h"

Haga correr el programa en la tarjeta (asegúrese de que la frecuencia de muestreo del códec está configurada a 48KHz), analice con el programa Scope los resultados obtenidos y verifique si el filtro cumple con las especificaciones. Para ello configure el generador de señales en modo de barrido (señal sinusoidal y 0,5V de amplitud) y observe lo que ocurre con el analizador de espectros (configúrelo en dB, auto escala y retención de pico).

Adjunte una captura de pantalla del resultado obtenido y exprese sus conclusiones.

Cuestión 2: Varíe poco a poco la amplitud de la señal de entrada hasta 1V y observe si ocurre algo inesperado. Explique lo que ocurre y a que puede deberse.

Cuestión 3: Repita la cuestión 1 pero ahora diseñe el filtro con una Fs de 24KHz.

Adjunte la captura de pantalla del resultado obtenido y exprese sus conclusiones.

Cuestión 4: La herramienta Fdatool permite varios métodos de diseño para un mismo filtro, a continuación diseñe un filtro paso bajo con las especificaciones que quiera utilizando los métodos de ventana (DesignMethod ->Window) que se han visto en la parte teórica de la asignatura, y compárelos entre ellos.

Pruebe primero a diseñarlos todos de orden 10 y luego repita la comparación cambiando el orden a 50.

Realice la misma experiencia diseñando otro tipo de filtro, por ejemplo un paso banda.

Adjunte las capturas de pantalla de la respuesta en frecuencia de los diferentes

filtros vistos con Scope y exprese sus conclusiones respecto a cómo influye el tipo de ventana y el orden en el diseño del filtro.

Cuestión 5: Por último, para conocer como se ha programado el filtro en C, vaya al programa principal y analice el código.

¿Qué papel desempeña la función "fir_convolution"? ¿Sería capaz de escribir su expresión matemática?

¿Para qué sirve la función "shift"? ¿Por qué es necesaria?

Opcional: A la vista del código, ¿qué ocurre si el filtro tiene más de 128 coeficientes? Haga las modificaciones que estime oportunas para que el comportamiento del filtro sea más preciso.

PRÁCTICA 4. FILTRADO IIR

OBJETIVOS

- Diseñar y caracterizar un filtro IIR con ayuda de Matlab, incluyendo el cálculo de los coeficientes.
- Exportar desde Matlab los coeficientes a CCS para su implementación.
- Interpretar los resultados obtenidos una vez que haya conseguido implementar el filtro diseñado en el DSP.
- Hacer un análisis de ventajas e inconvenientes de la implementación de un filtro IIR frente a uno FIR.

DESARROLLO

Para poder implementar un filtro FIR en nuestro DSP, lo primero que debe hacer es calcular los coeficientes del filtro. Para ello inicie Matlab y ejecute la herramienta FDATOOL.

Cuestión 1: Diseñe un filtro paso bajo tipo "IIR ChebyshevType I" con las siguientes especificaciones:

- Fs: 48 KHz
- Fpass: 4 KHz
- Fstop: 5 KHz
- Apass: 1 dB
- Astop: 80 dB

Una vez diseñado el filtro es necesario realizar un par de operaciones antes de poder exportar los filtros.

Fdatool por defecto calcula el filtro usando estructuras de segundo orden, pero para nuestra práctica es necesario convertir el filtro a una única sección, para ello, una vez diseñado el filtro vaya a: Edit ->Convert to Single Section

Ya está el filtro calculado en una única sección, ahora es necesario definir los parámetros de cuantificación, ya que en el diseño del filtro IIR van a resultar bastante críticos. Para configurar los parámetros de cuantificación diríjase a la

sección "Set quantizationparameters" pinchando en el icono resaltado en la imagen.

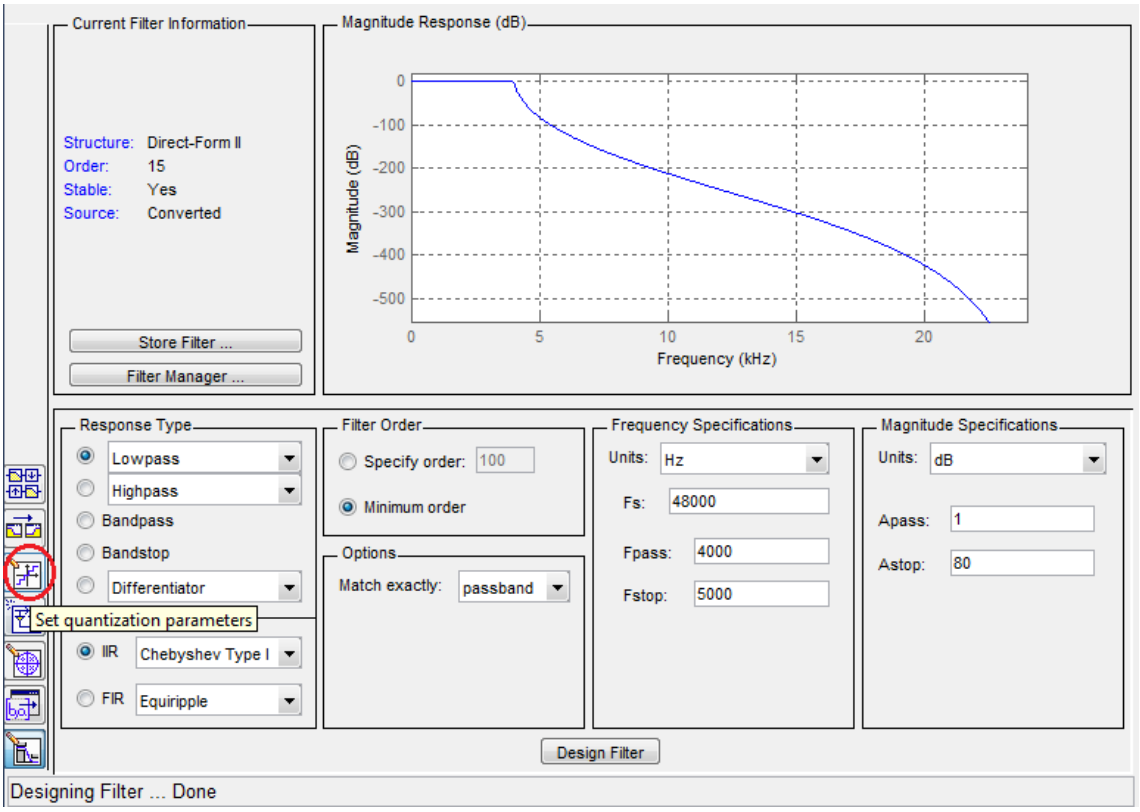

Ilustración 1. Configurar parámetros de cuantificación.

Una vez dentro, verá un menú donde deberá configurar la aritmética con la que se van a cuantificar los filtros. En este caso al estar trabajando con un DSP de punto fijo, deberá elegir "fixed-point" y aparecerá una ventana como la que sigue. Configure todos los campos, como se ve en las siguientes imágenes:

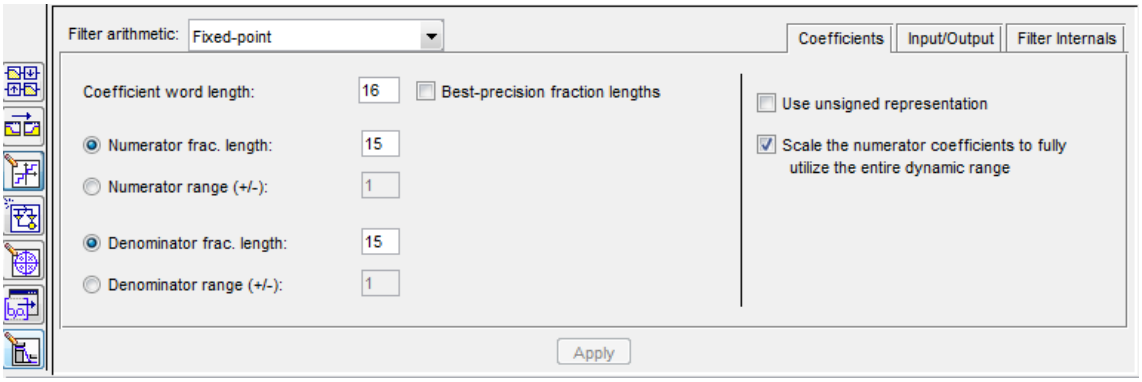

Ilustración 2. Configuración de parámetros de cuantificación I.

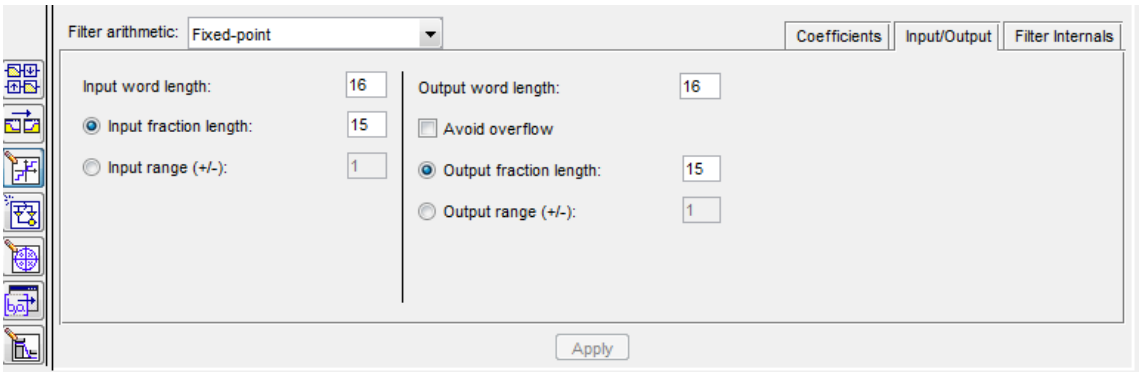

Ilustración 3. Configuración de parámetros de cuantificación II.

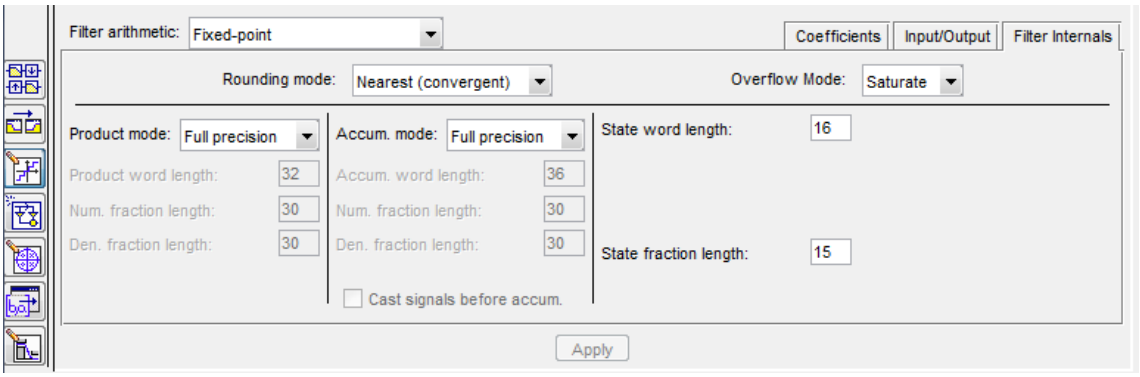

Ilustración 4. Configuración de parámetros de cuantificación III.

Una vez configurados pulse el botón "Apply" y genere el fichero de cabecera ".h" igual que se hizo en la práctica 3, pero esta vez guardándolo en el proyecto de la práctica 4.

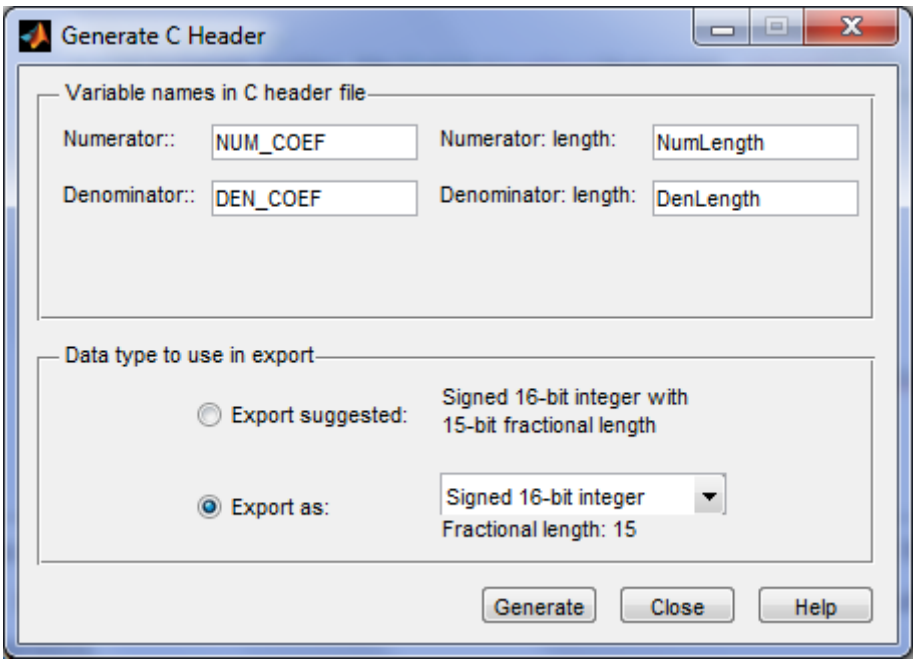

Ilustración 2. Definición de parámetros del fichero '.h'.

Como van a ser varios los ficheros ".h" que se van a generar se recomienda que escriba en el fichero, como comentario, las especificaciones del filtro.

Inicie CCS y seleccione el workspace de nombre “Practica 04” facilitado para esta práctica. Conecte la salida de audio del PC a la entrada de la tarjeta. Conecte la salida de la tarjeta a un la entrada de línea del PC.

Haga correr el programa y compruebe con Scope el funcionamiento del filtro. Configure la señal de entrada sinusoidal, con un barrido de frecuencias desde 0Hz hasta 20KHz y la mínima amplitud posible para evitar problemas de saturación. Puede ir variando la tensión de entrada para ver cómo se comporta el filtro.

Adjunte la captura de pantalla de la respuesta en frecuencia y anote sus conclusiones.

Cuestión 2: Repita el ejercicio anterior diseñando el mismo filtro pero ahora con el algoritmo de ChebyshevType II, Butterwoth y el Eliptico.

Adjunte las capturas de pantalla de la respuesta en frecuencia del filtro, compárelas entre ellas y con la anterior y anote sus conclusiones.

Cuestión 3: A la vista de los resultado obtenidos y comparándolos con los resultados de la práctica 3. ¿Qué conclusiones saca? ¿Qué ventajas e inconvenientes tiene cada uno de ellos?

Opcional 1: Para ver cómo se comporta el DSP ante los coeficientes sin cuantificar (aritmética de punto flotante), modifique el código del programa para que realice las operaciones con datos de tipo flotante en vez de enteros. Básicamente solo deberá cambiar algunas definiciones de tipos y eliminar la línea "`yn = yn>> 15;`" de la función "`iir_convolution()`".

Opcional 2: Averigüe que significa que el DSP empleado utilice un formato de datos de tipo Q15 y como puede afectar a la hora de realizar la cuantificación de los coeficientes del filtro.

PRÁCTICA 5. TRANSFORMADA DISCRETA DE FOURIER

OBJETIVOS

- Comprender el funcionamiento de la transformada discreta de Fourier en sistemas de tiempo discreto.
- Ser capaz de elegir un valor de ventana para el cálculo de la DFT acorde a la frecuencia de muestreo del códec y a la frecuencia de la señal de entrada.

CONOCIMIENTOS PREVIOS

La transformada discreta de Fourier (DFT) tiene un papel muy importante en el diseño, análisis y realización de sistemas y algoritmos de tratamiento de señales en tiempo discreto. Las propiedades básicas de la transformada de Fourier y de la DFT, hacen que analizar y diseñar sistemas en el dominio transformado sea conveniente y práctico. Este hecho, junto con la existencia de algoritmos eficientes para el cálculo explícito de la DFT, la convierte en un componente importante de muchas aplicaciones prácticas de los sistemas en tiempo discreto.

Definición de la DFT

Sea x_n una señal, la transformada discreta de Fourier (DFT) de N puntos se define:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}nk}, \qquad k = 0,1,\dots,N-1$$

Donde N es el numero de muestras de $X(\omega)$ en el intervalo $[0,2\pi]$, tomadas en las frecuencias

$$X_k = [X_0 X_1 \dots X_{N-1}]$$

La función X_k tiene periodo N, pues corresponde a una versión muestreada de la función periódica $X(\omega)$ que tiene periodo 2π.

El cómputo de cada valor X_k de la DFTevaluando directamente la ecuación analítica requiere N multiplicaciones complejas y $(N-1)$ sumas complejas.

Para obtener los N valores se necesitan en total N^2 multiplicaciones complejas y $N(N-1)$ sumas complejas. Como casi ningún procesador efectúa operaciones con números complejos, conviene expresar la ecuación como operaciones con números reales,

$$X_k = \sum_{n=0}^{N-1} \left[\left(Re\{x[n]\} Re\left\{e^{-j\frac{2\pi}{N}nk}\right\} - Im\{x[n]\} Im\left\{e^{-j\frac{2\pi}{N}nk}\right\} \right) - j\left(Re\{x[n]\} Im\left\{e^{-j\frac{2\pi}{N}nk}\right\} - Im\{x[n]\} Re\left\{e^{-j\frac{2\pi}{N}nk}\right\} \right) \right]$$

con $k = 0, 1, \ldots, N-1$, que demuestra que cada multiplicación compleja requiere cuatro multiplicaciones reales y dos sumas reales, y cada suma compleja requiere dos sumas reales.

Por tanto, para cada valor de k , el cálculo directo de X_k necesita $4N$ multiplicaciones reales y $(4N-2)$ sumas reales. Como X_k se debe calcular para N valores diferentes de k, el cálculo directo de la DFT de una sucesión $x[n]$ requiere $4N^2$ multiplicaciones reales y $N(4N-2)$ sumas reales.

Como el número de cálculos y, en consecuencia, el tiempo de cómputo es aproximadamente proporcional a N^2, el número de operaciones aritméticas necesarias para calcular la DFT por el método directo es muy elevado para valores grandes de N.

Es por esto que es de suma importancia elegir un valor correcto de N

DESARROLLO

Para el desarrollo de esta práctica se va ha implementado la DFT a partir de la siguiente ecuación,

$$X(k) = \sum_{n=0}^{N-1} x(n) cos\left(\frac{2\pi kn}{N}\right) - j \sum_{n=0}^{N-1} x(n) sen\left(\frac{2\pi kn}{N}\right)$$

con$k = 0,1, \ldots, N-1$.

Inicie CCS y seleccione el workspace de nombre “Practica 05” facilitado para esta práctica. Conecte la salida de audio del PC a la entrada de la tarjeta.

Conecte la salida de la tarjeta a un la entrada de línea del PC.

$$X(k) = \sum_{n=0}^{N-1} x(n) cos\left(\frac{2\pi kn}{N}\right) - j \sum_{n=0}^{N-1} x(n) sen\left(\frac{2\pi kn}{N}\right)$$

Cuestión 1: Examine el código del programa identificando las principales sentencias y los valores que determinan el cálculo de la DFT. ¿Qué tamaño de ventana tiene la DFT tal y como se ha implementado?

Cuestión 2: Para visualizar los resultados de la DFT se van a configurar las graficas del depurador ya que con el Scope, aunque se podrá ver también la señal, no es fácil analizar los resultados. Por tanto será necesario configurar la herramienta de gráficas como se hizo en la práctica 1.

Coloque un break-point antes de la función de escritura y visualice el canal derecho en la gráfica. Configure una señal sinusoidal con Scopey varíe su frecuencia lentamente desde 0 hasta el máximo posible, con una amplitud de 0,2V.

Active la opción Run/Clock/Enable de CCS. Esto debería mostrar el elemento  en la barra de estado. Este módulo cuenta los ciclos de reloj que ha contado el emulador después de cada punto de ruptura. Este contador es incremental, pero si se hace doble clock sobre el icono del reloj, el contador vuelve a cero. De esta forma se puede evaluar el coste computacional del código que se ha implementado.

A continuación repita el ejercicio cambiando el tamaño de BUFFER_SIZE (N) primero a 64 y luego a 1024, rellene la siguiente tabla y exprese sus conclusiones. Si lo desea se puede apoyar en alguna captura de pantalla.

Tamaño de ventana	Ciclos de reloj
128	
64	
1024	

Cuestión 3: ¿Cómo afecta al cálculo de la DFT la frecuencia de muestreo? ¿Qué ventajas e inconvenientes puede implicar configurar una frecuencia de muestreo más pequeña?